TAMPEREEN TEKNILLINEN YLIOPISTO

ANSSI KUUTTI

MUSIIKIN TILAUS- JA HALLINTAJÄRJESTELMÄ

Diplomityö

Tarkastajat: Jari Peltonen
ja Tommi Mikkonen
Tarkastajat ja aihe hyväksytty
Tietotekniikan osastoneuvoston
kokouksessa 08.06.2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
KUUTTI, ANSSI: Musiikin tilaus- ja hallintajärjestelmä
Diplomityö, 50 sivua
Marraskuu 2011
Pääaine: Ohjelmistotuotanto
Tarkastajat: Yliassistentti Jari Peltonen, Professori Tommi Mikkonen, DI Pasi Kovanen
Avainsanat: taustamusiikkipalvelut, web-sovellus, ekstranet, Java EE

Taustamusiikkia soitetaan monenlaisissa tiloissa, kuten yökerhoissa, ravintoloissa ja ostoskeskuksissa. Yleensä musiikin toistamiseen käytetään jotakin järjestelmää, jolla on helppo ajastaa tietty musiikki soimaan tiettyllä hetkellä. Tällaista järjestelmää kutsutaan taustamusiikkisoittimeksi. Suomen ensimmäisen taustamusiikkisoittimen kehittänyt yritys halusi tarjota asiakkailleen musiikin tilaus- ja hallintajärjestelmän, jolla yrityksen asiakkaat voisivat ylläpitää mediakirjastojaan ja asiakkuustietojaan. Tarvittavia toimintoja olivat muun muassa uuden musiikin tilaus, soittolistojen laatiminen ja soittolistojen ajastus.

Tässä diplomityössä suunniteltiin ja toteutettiin musiikin tilaus- ja hallintajärjestelmä yhdessä asiakasyrityksen kanssa. Asiakkaalla oli käytössä aiemmin itse toteuttamansa järjestelmän ensimmäisen version. Ensimmäisestä versiosta saadun palautteen ja kehitysideoiden pohjalta asiakas määritteli uuden version toiminnot ja laati alustavat käyttöliittymäsuunnitelmat. Uuden version toteutus tehtiin iteratiivisesti ja asiakas oli tiiviisti mukana järjestelmän toteutuksessa ja testauksessa.

Järjestelmän toteutuksessa panostettiin automatisoituun testaukseen, jatkuvaan integrointiin ja käyttöliittymäsuunnitteluun. Järjestelmästä tulikin visuaalisesti näyttävä ja helppokäyttöinen. Asiakkaan antama palaute järjestelmästä ja koko prosessista on ollut erittäin positiivista. Järjestelmän käyttöönotto oli kirjoitushetkellä vielä kesken, joten loppukäyttäjiltä ei ole saatu palautetta järjestelmästä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
KUUTTI, ANSSI: Music ordering and management system
Master of Science Thesis, 50 pages
November 2011
Major: Software engineering
Examiners: Assistant Professor Jari Peltonen, Professor Tommi Mikkonen, M.Sc Pasi Kovanen
Keywords: background music services, web application, extranet, Java EE

Background music is played in various places such as nightclubs, restaurants and shopping malls. Usually the background music is played using software systems that enable easy playback of certain tracks at given moment using scheduled playlists. These kind of software systems are called background music players. A company responsible for developing Finland's first background music player decided to offer a music ordering and management system to their customers. The system was designed to help the customers update and maintain their media libraries and account info. Other desired features were music ordering, playlist creation, and scheduling of playlists.

The goal of this thesis was to design and develop a music ordering and management system in collaboration with the client. The client had developed the first version of the system formerly. Based on the feedback from the first version of the system, the client had defined the features and functions for the new system. The client had also done some preliminary user interface designs on some of the features. The system implementation was carried out iteratively and the client participated in designing and testing of the system.

Automated testing, continuous integration, and user interface design were given top priority during development. The result is visually attractive and easy to use. Feedback given by the client was very positive during the whole process. At the moment of writing, the system was still waiting to be deployed, so no real life end-user feedback has been received.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston Ohjelmistotekniikan laitoksella. Aloitin työn tekemisen syksyllä 2010 ja kirjoittamisen keväällä 2011. Lopullisen viimeistelyn työn kirjalliseen asuun tein syksyllä 2011.

Tein työn osana Vincit Oy:ssä tekemääni asiakasprojektia. Olen työskennellyt Vincitissä keväästä 2010 saakka. Vincit on suomalainen innokas ohjelmistotalo, joka työllistää kirjoitushetkellä noin 55 ohjelmistoalan ammattilaista Tampereella, Hervannassa. Vincit on tulorahoitteinen, kannattava kasvuyritys, joka on perustettu vuonna 2007. Ohjelmistokehityksessä Vincitin vahvuuksia ovat mobiiliohjelmistot, sulautetut ohjelmistot ja tietojärjestelmät.

Työtäni ohjasi Vincitin puolelta DI Pasi Kovanen. Laitokselta työtäni tarkasti ja ohjasi yliassistentti Jari Peltonen ja professori Tommi Mikkonen. Lämpimät kiitokset heille kaikille. Kiitokset myös puolisolleni Anna Kurkiselle tuesta ja ymmärtämisestä.

Tampereella 12. lokakuuta 2011

Anssi Kuutti
Puh. 050 530 7938
anssi@kuutti.eu

SISÄLLYS

KUVAT

1. JOHDANTO

Taustamusiikkipalvelut ovat kaikkien käyttämä, mutta vähemmän tunnettu musiikkialan palvelu. Taustamusiikkia soitetaan ravintoloissa, yökerhoissa, pubeissa, kauppakeskuksissa ja monissa muissa julkisissa paikoissa. Tämän diplomityön aiheena on musiikin tilaus- ja hallintajärjestelmä, jolla taustamusiikkipalveluita tarvitseva yritys voi hallita mediakirjastonsa sisältöä ja ylläpitää asiakkuustietojaan. Mediakirjasto voi sisältää esimerkiksi yökerhon musiikin, musiikkivideot, karaokevideot ja mainokset joita yökerhossa asiakkaille toistetaan. Mediasta voi laatia monipuolisia listoja, joita voidaan ajastaa soimaan tiettyyn aikaan tietyssä tilassa, esimerkiksi yökerhon tanssilattialla. Järjestelmä on tehty Vincit Oy:n toimesta alihankintana eräälle asiakkaalle ja siitä tulee osa asiakkaan laajempaa palvelukokonaisuutta. Asiakas on Pohjoismaiden suurin taustamusiikkipalveluita tuottava yritys, joka on toimittanut taustamusiikkisoitinta ja muita palveluita jo lähes kahdenkymmenen vuoden ajan suomalaisille yrityksille.

Järjestelmän toteutus aloitettiin kesällä 2010 ja kirjoitushetkellä, syksyllä 2011, on projektissa meneillään käyttöönottovaihe. Toteutus tehtiin kahden viikon iteraatioissa Java EE -ohjelmistoalustalle. Sovelluksen laadun parantamiseksi toteutuksessa käytettiin jatkuvaa integrointia, useita käytettävyystestauksen menetelmiä ja automatisoituja testejä. Järjestelmälle laadittiin yksikkö- ja järjestelmätestejä toteutustyön ohella, jotka automatisoidusti ajettiin integrointi- ja kehitysympäristöissä.

Järjestelmän toteutukseen on osallistunut useita henkilöitä sekä asiakkaan että Vincitin puolelta. Projektin päävastuu on ollut Vincitillä projektinhallinnasta vastanneella henkilöllä. Työn kirjoittaja on ollut vastuussa järjestelmän teknisestä toteutuksesta. Kirjoittaja on myös osallistunut määrittelyiden tarkennuksiin, järjestelmän suunnitteluun, toteutukseen, testaukseen ja antanut tukea käyttöönottoon liittyvissä seikoissa. Diplomityö on tarkoitettu projektin asiakkaalle ja web-palvelun suunnittelusta, toteuttamisesta ja testaamisesta kiinnostuneille, sekä ennen kaikkea työn kirjoittajan opinnäytetyöksi.

Luku 2 kuvaa projektin kannalta oleelliset WWW-tekniikat. Luvussa käydään läpi millaisia tekniikoita WWW-ympäristössä julkaistuissa sovelluksissa yleisesti käytetään, sekä mitkä ovat web-sovelluksille yleisiä arkkitehtuurisia piirteitä. Lopuksi tutustutaan web-sovelluksen testauksessa, jatkuvassa integroinnissa ja laadunhallinnassa käytettyihin tekniikoihin.

Luku 3 käy läpi projektin taustat ja edellytykset. Aluksi kuvataan projektin tilannut asiakas, asiakkaan liiketoiminta, liiketoiminnan edellytykset ja mistä tarve uudelle järjestelmälle on syntynyt. Luvussa käydään läpi myös asiakkaan palvelukokonaisuudet ja muut järjestelmät, sekä kuinka ne liittyvät projektissa toteutettuun järjestelmään. Lisäksi luvussa käydään läpi järjestelmälle asetetut vaatimukset, suunnitteluperiaatteet ja kuvataan vaatimukset täyttävä ohjelmisto.

Luku 4 kuvaa järjestelmän toteutuksen. Aluksi kuvataan järjestelmän korkean tason arkkitehtuuri ja sovelluksen ohjelmistoarkkitehtuuri, sekä arkkitehtuurin taustalla olevat vaatimukset ja periaatteet. Seuraavaksi kuvataan järjestelmän suunnittelu, keskeisten toimintojen toteutus ja ohjelmiston testaus. Lisäksi luvussa käydään läpi joitain käyttöönottoon ja ylläpitoon liittyviä seikkoja.

Luvussa 5 arvioidaan projektin onnistumista. Aluksi kuvataan mikä on projektin tila kirjoitushetkellä, eli missä vaiheessa projektin elinkaarta ollaan, mitä on tehty ja mitä on tekemättä. Seuraavaksi arvioidaan kuinka projekti vastaa sille asetettuja vaatimuksia ja millainen on järjestelmän tekninen laatu. Luvussa käydään läpi myös projektin aikana ja lopuksi asiakkaalta saatua palautetta. Luvun lopussa on myös joitain kirjoittajan ajatuksia projektin jatkokehityksestä ja tulevaisuudesta.

Luku 6 on yhteenveto diplomityöstä. Luvussa käydään läpi työlle asetetut tavoitteet, mitä työssä tehtiin ja mitkä ovat työn tulokset.

2. WWW-TEKNIIKAT

Tässä luvussa kuvataan WWW-hypertekstijärjestelmässä julkaistuissa sovelluksissa yleisesti käytettyjä tekniikoita. Ensin tarkastellaan yleisesti web-sovelluksen erityispiirteitä verrattuna tavallisiin työpöytäsovelluksiin. Tarkastelemme myös kuinka web-sovelluksen arkkitehtuuri ja sen valinta vaikuttavat teknologiavalintoihin ja käytettyihin suunnittelumalleihin. Seuraavaksi esitellään palvelin- ja asiakaspäässä käytettäviä tekniikoita. Tärkeä osa web-sovelluksien toteutusta ovat myös testauksen, jatkuvan integroinnin ja laadunhallinnan tekniikat, joita tarkastellaan lopuksi.

2.1 Web-sovelluksen erityispiirteitä

Internet ei ole enää pelkästään staattisten sivustojen verkosto, vaan media, jonka avulla palveluita voidaan tarjota tietoturvallisesti ja kustannustehokkaasti. Palvelun toimintalogiikka voidaan keskittää palvelimelle, eikä loppukäyttäjien tarvitse asentaa päätelaitteilleen selaimen lisäksi muita ohjelmistoja. Päivitykset voidaan ottaa käyttöön ilman, että käyttäjien tarvitsee tehdä mitään. Lisäksi palvelun käyttö on mahdollista mistä ja milloin tahansa.

Web-sovelluksen arkkitehtuuri poikkeaa yleensä tavallisen työpöytäsovelluksen arkkitehtuurista, sillä käyttöliittymä on käyttäjän selaimessa ja itse sovellus palvelimella. Tiedonsiirto selaimen ja palvelimen välillä tapahtuu HTTP-protokollaa [32] käyttäen TCP/IP-protokollapinon päällä. HTTP:n heikkous web-sovelluksen näkökulmasta on sen tilattomuus. Palvelin ei voi erottaa eri asiakkailta tulleita pyyntöjä toisistaan käytettäessä tilatonta protokollaa ilman erillistä mekanismia tilanhallintaan. Ongelma voidaan kuitenkin kiertää esimerkiksi käyttämällä evästeitä (cookies), jotka tallentuvat asiakaspäähän selaimeen ja jotka välitetään pyyntöjen mukana asiakkaan ja palvelimen välillä. Toinen paljon käytetty tapa on palvelinpään istunnot, jossa palvelin ylläpitää tilatietoja välittäen HTTP-pyyntöihin piilotettua istuntotunnistetta.

Käyttöliittymän esittäminen käyttäjän selaimessa pitää web-palvelun käytön yksinkertaisena; käyttäjän ei tarvitse asentaa tietokoneelle sovellusta, jotta hän voi käyttää palvelua. Tästä seuraa kuitenkin myös WWW-ympäristön isoin ongelma, sillä usein eri selaimet esittävät saman web-sivun hieman eri tavalla. Yleensä erot ovat kosmeettisia ja harmittomia, mutta ilman huolellista testaamista selainten väliset erot saattavat estää palvelun käytön kokonaan joltain käyttäjäryhmältä. Selain-

valmistajien välinen kova kilpailu ja uudet standardit parantavat tilannetta kuitenkin koko ajan.

2.1.1 Kerrosarkkitehtuuri web-sovelluksessa

Kerrosarkkitehtuuri on yleisesti käytetty asiakas-palvelin-arkkitehtuuri, jossa sovelluksen toiminta jakautuu sovelluskerroksille. Alempi kerros toimii palvelimena ylemmälle kerrokselle ja kullakin kerroksella on selkeästi rajattu vastuualue. Web-sovelluksissa monesti käytetty arkkitehtuuri on kolmikerrosarkkitehtuuri, jossa on esitystapakerros, liiketoimintakerros ja tietovarastokerros. [25]

Ylin kerros on esitystapakerros, jonka vastuulla on käyttöliittymän tarjoaminen käyttäjälle ja vuorovaikutustapahtumien välittäminen alemmalle kerrokselle. Liiketoimintakerros vastaa sovelluksen varsinaisesta toimintalogiikasta ja toimii palvelimena esitystapakerrokselle. Se reagoi esitystapakerroksen tuottamiin tapahtumiin, päivittää esitystapakerrosta sekä noutaa ja jalostaa tietovarastokerroksesta haettua tietoa. Alin kerros, eli tietovarastokerros, abstrahoi sovelluksen tietovaraston ja toimii palvelimena liiketoimintakerrokselle. Sen vastuulla on huolehtia tiedon pysyvyydestä ja saavutettavuudesta tarjoamalla tietopalvelut liiketoimintakerroksesta riippumattomalla tavalla. [25]

Vastuiden jakaminen kerroksittain parantaa sovelluksen hallittavuutta ja laajennettavuutta. Kerrosmalleille yleinen piirre on mahdollisuus hajauttaa kerrokset verkon yli. Web-sovelluksessa käyttöliittymä esitetään asiakkaan selaimessa, eli aina selkeästi verkkoyhteyden päässä, mutta myös tietovarastokerros ja liiketoimintakerros ovat usein eri palvelimilla.

2.1.2 MVC-suunnittelumalli

MVC (model-view-controller eli malli-näkymä-käsittelijä) on paljon käytetty arkkitehtuurimalli, joka yksinkertaisella vastuujaolla pyrkii erottamaan käyttöliittymän sovelluksen logiikasta. Arkkitehtuurimallin perusajatus on sovelluslogiikan uudelleenkäytettävyys toisessa esitysmuodossa. Lisäksi arkkitehtuurimalli helpottaa käyttöliittymän ja sovelluksen tilan ja datan synkronoimista keskenään. [25]

MVC-arkkitehtuurimallia noudattava järjestelmä jakautuu osiin, jotka ovat kolmea loogisesti erilaista tyyppiä. Mallit ovat osia sovelluksen datasta tai tilasta. Näkymät ovat yksittäisiä käyttöliittymäkomponentteja tai kokonaisia käyttöliittymiä. Käsittelijät toimivat mallien ja näkymien välissä eräänlaisina sovittimina pitäen käyttöliittymät ja sovelluksen datan ajan tasalla. [25]

2.1.3 DAO-suunnittelumalli

DAO (Data Access Object) on suunnittelumalli, joka tarjoaa abstraktin rajapinnan tietovarastokerrokselle. Kukin DAO-luokka tarjoaa palvelut tietyn malli-luokan olioiden hakemiseen, luomiseen, päivittämiseen ja poistamiseen. Suunnittelumalli ohjaa kirjoittamaan selkeästi tiedon käsittelyyn liittyvän logiikan erilleen muusta sovelluksen ohjelmakoodista. Piilottamalla varsinaisen tietokannan palvelut DAO-rajapintojen taakse tietokannanhallintajärjestelmä on helposti vaihdettavissa, ja tarvittaessa erilaiset välimuistit voidaan ottaa käyttöön skaalautumisen parantamiseksi. DAO-luokat on myös helppo alustaa ja tarjota käyttöön siellä, missä tietovarastopalveluita tarvitaan. [3]

2.2 Palvelinpään tekniikat

Tässä kohdassa kuvataan web-sovelluksen asiakas-palvelin-arkkitehtuurin palvelinpäässä käytetyt tekniikat. Palvelin ei välttämättä tarkoita yksittäistä palvelinta vaan saattaa todellisuudessa koostua useista eri palvelimista.

2.2.1 Java EE

Java on Sun Microsystemsin kehittämä ja nykyisin Oraclen omistama teknologiaperhe. Se koostuu tulkattavasta oliopohjaisesta ohjelmointikielestä (Java), ajoaikaisesta ympäristöstä (JRE, Java Runtime Environment) ja virtuaalikoneesta (JVM, Java Virtual Machine). Java on julkaistu GPL-lisenssin [10] alaisena avoimena lähdekoodina. [15]

Java EE (Java Platform, Enterprise Edition) on paljon käytetty ohjelmistoalusta palvelinsovelluksen toteutuksessa. Alustasta on olemassa myös suppeampi perus versio (Standard Edition), josta puuttuvat palvelinympäristöjä varten varatut kirjastot. Käytännössä Java EE -standardi mahdollistaa web-sovelluksen kehittämisen käyttäen standardikirjastoja niin, että sovellus voidaan helposti ottaa käyttöön missä tahansa standardin toteuttavassa sovelluspalvelimessa. Useilla ohjelmistovalmistajilla on omat sovelluspalvelimensa, jotka toteuttavat Java EE -ohjelmistoalustastandardin. Sovellus voidaan esimerkiksi kehitysvaiheessa tehdä käyttäen kevyttä Jetty web-palvelinta [20] ja lopulta sijoittaa Apache Tomcat -sovelluspalvelimelle [1] ilman, että mitään erityisiä konfigurointeja tarvitsee tehdä.

Sovelluspalvelimet tarjoavat web-säiliön (servlet container), johon voidaan sijoittaa Java Servlet API -rajapinnan toteuttavia luokkia, jotka sovelluspalvelin julkaisee palveluiksi. Web-säiliö jakaa käyttäjän selaimen lähettämät HTTP-pyynnöt servleteille paketoimalla ne HttpRequest-luokan instanssiin. Servletit käsittelevät pyynnöt ja vastaavat niihin palauttamalla HttpResponse-luokan instanssin, jonka web-säiliö

välittää jälleen käyttäjälle HTTP-pakettina.

Muita sovelluspalvelimien yleisesti tarjoamia palveluita ovat JavaServer Pages -sivujen (JSP) tulkkaus, käyttäjä istunnot ja tietokantayhteydet. JavaServer Pages on teknologia, joka helpottaa HTML-sivujen (katso alakohta 2.3.1) dynaamista, ajonaikaista luontia. Arkkitehtuurin kannalta ajateltuna JSP-sivut ovat korkeantason servlet-luokkia, sillä ennen varsinaista tavukoodiksi kääntämistä, JSP-sivut käännetään servlet-luokiksi, jotka sovelluspalvelin yleensä tallentaa levylle [15]. JSP-sivujen ohjelmoinnissa voidaan käyttää Expression Language -nimistä skriptikieltä [16], joka on osa Java-ohjelmistoalustaa tai erityisten tagien sisällä myös Java-ohjelmakoodia. JSP:n tagikirjastoja voi myös laajentaa tuomalla sivulle omia tagikirjastoja (TagLib). [15]

Web-säiliöt tarjoavat HttpSession-luokkaan paketoidun istunnon, jolloin kunkin käyttäjän HTTP-pyynnöt saadaan liitettyä tiettyyn HttpSession-luokan instanssiin. Liittäminen tehdään käyttäen evästeitä (cookies) tai HTTP:n otsikkokentissä kuljetettavia istuntotunnisteita, riippuen sovelluspalvelimesta ja käyttäjän selaimesta. Myös tietokantayhteydet voidaan kuvata sovellus- tai sovelluspalvelinkohtaisesti JDBC-rajapinnan (Java Database Connectivity) avulla.

2.2.2 Relaatiotietokannat ja olio-relaatio-muunnos

Relaatiomalli on rakennetason tietomalli, jota käytetään relaatiotietokannoissa. Relaatiomallissa tieto esitetään relaatioina, joiden ilmentymiä kutsutaan tauluiksi. Relaatiot muodostuvat monikoista ja ominaisuuksista. Taulukkoon verrattaessa monikot ovat rivejä ja ominaisuudet sarakkeita. [2]

Relaatiomallissa sovelluksen tietomalli pyritään suunnittelemaan siten, että jokaista sovelluksen käsitettä vastaa yksi relaatio. Relaatioille asetetaan monikot yksilöivä pääavain. Käsitteiden väliset suhteet esitetään mallissa käyttäen vierasavain-viittauksia. [2]

Relaatiotietokantaa käytetään sovelluksessa erillistä tietokannanhallintajärjestelmää hyödyntäen. Sovellukset keskustelevat tietokannanhallintajärjestelmän kanssa käyttäen standardoitua SQL-kieltä (Structured Query Language). IBM kehitti SQL:n samaan aikaan ensimmäisen relaatiotietokannan kanssa, monet tietokantajärjestelmien valmistajat ovat kuitenkin tehneet omia laajennuksia kieleen. [31]

Relaatiomallin yksinkertaisuus on tehnyt relaatiotietokannoista hyvin suosittuja, vaikka sovellukset harvoin käsittelevät tietoa relaatioiden kautta. Oliokieliä käytettäessä relaatiot täytyy muuttaa olioiksi, jotta relaatiotietokannasta haettua tietoa voidaan hyödyntää sovelluksessa. Tätä kutsutaan olio-relaatio-muunnokseksi. Muunnoksen tekeviä kirjastoja on saatavilla lähes kaikille oliokielille, ja esimerkiksi Javalle niitä löytyy lukuisia. Relaatiomalli ja tietokannanhallintajärjestelmän käyttö helpottaa yhden tietovaraston jakamista usean sovelluksen kesken selkeyttämällä

vastuujakoa.

2.2.3 JPA ja EclipseLink

JPA 2.0 (Java Persistence API) on olio-relaatio-muunnokseen tarkoitettu rajapinta, joka on osa Oracle EJB 3.0 -spesifikaatiota (Enterprise Java Beans). Rajapintaa käyttäen relaatiotietokannan monikot voidaan muuttaa sovelluksen käyttämiksi entiteetti olioiksi. Tämä mahdollistaa varsinaisen tietokannan ja tietokannanhallintajärjestelmän kätkemisen JPA-rajapinnan taakse. JPA:sta löytyy myös työkalut transaktioiden ja välimuistin hallintaan. [24]

JPA-spesifikaatio sisältää JPQL-kyselykielen (Java Persistence Query Language), joka mahdollistaa tiedon monipuolisen hakemisen tietokannasta. JPA-toteutus jäsentää kyselykielellä kuvatun kyselyn ja tekee varsinaisen tietokantakyselyn tietokannanhallintajärjestelmään esimerkiksi JDBC-rajapintaa käyttäen. Haun tuloksista luodaan tämän jälkeen sovelluksen tuntemia olioita, jotka palautetaan alkuperäisen kyselyn tekijälle. Monet JPA-toteutukset tukevat lisäksi tietokantaskeeman luontia, mikä mahdollistaa tietokantataulujen luonnin ohjelmakoodin perusteella. Tämä nopeuttaa sovelluksen kehitystä ja toisaalta kehitysvaiheessa on helppo kokeilla eri tietokantoja tai esimerkiksi automatisoitu testaus voidaan tehdä lennossa luotua tietokantaa vasten. [24]

EclipseLink on avoimen lähdekoodin ohjelmistokehys, joka toteuttaa muun muassa JPA-rajapinnan [6]. JPA-rajapinnan toteuttavia kirjastoja on lukuisia, mutta näistä EclipseLink on yksi suosituimmista kaksitasoisen laajan välimuistin ja tarkan spesifikaatiossa pysymisen johdosta. Sun Microsystems onkin valinnut EclipseLink-projektin JPA 2.0 -spesifikaation referenssitoteutukseksi [6]. Kun valitsee jonkin rajapinnan toteutuksen spesifikaatiossa pysymisen perusteella, voi myöhemmin tarpeen vaatiessa kirjaston vaihtaa toiseen ilman, että ohjelmakoodiin tarvitsee tehdä mittavia muutoksia. Tällaisille kirjastoille on myös paremmin ohjeita ja esimerkkikoodia saatavilla.

2.2.4 Stripes-sovelluskehys

Stripes on toimintopohjainen sovelluskehys, jonka keskeisin etu on esitystapakerroksen yksinkertaistuminen. Toimintopohjaisuus tarkoittaa sitä, että sovellus rakentuu yksittäisistä toiminnoista. Stripes kutsuu toimintoja ActionBean:ksi, jotka ovat saman nimisen rajapinnan toteuttavia Java-luokkia. Kukin ActionBean kytketään yksittäiseen URL:ään (Uniform Resource Locator), ja palvelua selaileva käyttäjä ohjataan näin toiminnosta toiseen käyttäen perinteisiä verkkolinkkejä. Kun jokainen toiminto on kytketty omaan URL:ään, voivat loppukäyttäjät selata sovellusta selaimen takaisin ja eteenpäin -toimintoja hyödyntäen, sekä tallentaa sivuja selaimen

kirjanmerkeiksi. [38]

Stripesin ActionBean-luokat ovat MVC-mallin (kohta 2.1.2) mukaisia käsittelijöitä. Suuri osa sovelluksen liiketoimintalogiikasta sisältyy käsittelijöihin, mutta käsittelijöitä itsessään ei sidota käyttäjien istuntoihin. Käsittelijät ja käyttäjien istunnot liitetään toisiinsa käyttäen ActionBeanContext-luokkaa. Näkymät ovat Stripesin tapauksessa yleensä JSP-tiedostoja (alakohta 2.2.1). Käsittelijät tarjoavat palveluita näkymille epäsuorasti, eli JSP-sivut voivat kutsua osajoukkoa ActionBean-luokkien julkisista metodeista. Stripes sisältää oman tagikirjaston (Stripes TagLib), joka helpottaa näkymien ohjelmointia ja näkymän ja ActionBeanin välistä vuorovaikutusta.

Jo ensimmäisistä Java EE -spesifikaatioon perustuvista web-sovelluskehyksistä lähtien kehittäjät ovat joutuneet sietämään yhä lisääntyvää XML-konfiguroinnin määrää. Stripesissa konfiguroinnin määrä pyritään pitämään minimissä muun muassa käyttäen nimeämiskäytäntöjä konfiguroinnin sijasta. Esimerkiksi luokka tunnistetaan ActionBeaniksi, jos sen nimi päättyy merkkijonoon ActionBean. Erillisten konfiguraatiotiedostojen tarvetta vähentää myös asetuksien asettaminen ohjelmakoodin seassa käyttäen Java 1.5:n mukana tulleita annotaatioita. Annotaatiot ovat eräänlaista metatietoa ohjelmakoodista. Stripesin hyvä puoli on myöskin sen keveys; projektin saa pystyyn hyvin pienellä vaivalla, eikä kaikkia ominaisuuksia ole pakko ottaa käyttöön ennen kuin niitä tarvitsee. [38]

2.2.5 Spring-sovelluskehys

Spring on laaja avoimen lähdekoodin sovelluskehys Java ohjelmistoalustalle [37]. Kehyksen ensimmäisen version on kirjoittanut Rod Johnson, joka julkaisi kehyksen kirjansa *Expert One-on-One J2EE Design and Development* yhteydessä vuonna 2002 [21]. Tämän jälkeen kehys on saavuttanut valtavan suosion ja voittanut useita palkintoja [22, 18].

Kehys sisältää useita moduuleita, kuten IoC-säiliön sekä tiedon- ja transaktioidenhallintamoduulit. IoC (Inversion of Control) on arkkitehtuurityyli (joissain yhteyksissä kutsutaan myös suunnittelumalliksi), jonka keskeisin ajatus on vastuun päinvastainen jakaminen kuin perinteisesti proseduraalisessa ohjelmoinnissa on totuttu [9]. Perinteisesti ohjelmakoodissa on otettu kirjasto käyttöön kutsumalla jotain kirjaston tarjoamaa funktiota, joka suorittaa tietyn tehtävän. Käänteisesti tämä voitaisiin toteuttaa niin, että kirjastolle tarjotaan takaisinkutsurajapintaa käyttäen liityntä funktion mahdollisiin jälkitoimenpiteisiin. Tämän jälkeen kontrolli annetaan kirjastolle, joka huolehtii funktion kutsumisesta ja jatkotoimenpiteistä. Käänteistä vastuuta kutsutaan myös Hollywood-periaatteeksi; *älä soita meille, me soitamme teille*. [25]

Springin IoC-säiliölle voidaan osoittaa esimerkiksi Java-paketti, joka sisältää komponentit, jotka halutaan säilöä ja tämän jälkeen nuo komponentit ovat käytettävissä

missä vain käyttäen riippuvuuksien injektiota (Dependency Injection). Kehys huolehtii tällöin komponentin alustamisesta ja instanssin tarjoamisesta sitä tarvitsevalle osapuolelle. Säiliöllä voidaan karsia täysin riippuvuudet rajapintojen käyttäjien ja toteuttajien väliltä, eikä käyttäjän tarvitse kirjoittaa kontrollikoodia, sillä vastuu injektioista on annettu Spring-kehykselle. [37]

Tiedonhallintamoduuli mahdollistaa Springin yhteiskäytön JPA-toteutuksen, kuten EclipseLinkin kanssa, toteuttamalla JPA-säiliöille vaaditun SPI (Service Provider Interface) -rajapinnan. Moduuli sisältää muun muassa resurssienhallinnan, poikkeuksienhallinnan ja transaktiotuen. Transaktioidenhallintamoduulia voidaan käyttää yhdessä tiedonhallintamoduulin kanssa, jolloin halutessaan transaktiot voi ulkoistaa täysin Spring-kehyksen vastuulle. [37]

2.3 Asiakaspään tekniikat

Tässä kohdassa kuvataan web-sovelluksen asiakas-palvelin-arkkitehtuurin asiakaspäässä käytetyt tekniikat. Asiakkaan ohjelmistoympäristö on yleensä selain, joka tukee kohdassa kuvattuja tekniikoita. Tekniikat ovat pääasiassa esitystapatekniikoita, mutta asiakaspäässä saatetaan myös toteuttaa liiketoimintalogiikkaa.

2.3.1 HTML

HTML (HyperText Markup Language) on rakenteinen merkkauskieli, jolla websivuja kuvataan. Kieli koostuu elementeistä, joilla kullakin oma erityinen semanttinen tai visuaalinen merkityksensä. Elementit voivat sisältää ominaisuuksia, eli attribuutteja, tyyliasetuksia (katso kohta 2.3.2), sekä komentosarjakoodia. Ominaisuudet voivat olla esimerkiksi linkin kohde tai kuvan reunuksen paksuus. Elementtiin upotetulla komentosarjakoodilla voidaan kuvata elementille esimerkiksi tapahtumankäsittelijöitä. Tapahtumat voivat olla esimerkiksi hiirellä osoitus tai fokuksen siirtyminen. Komentosarjakoodi voi olla esimerkiksi JavaScript-koodia, josta lisää kohdassa 2.3.3. [12]

HTML välitetään palvelimelta asiakkaan selaimelle yleensä HTTP-protokollaa käyttäen. Selaimet lukevat web-sivun HTML-koodin ja jäsentävät sen visuaaliseksi ja auditiiviseksi esitykseksi. HTML ja HTTP mahdollistavat myös vuorovaikutuksen toiseen suuntaan. Esimerkiksi kaavake-elementin (form) sisältämät tiedot voidaan lähettää palvelimelle HTTP-paketissa. [12]

Kielen uusin virallinen versio on HTML 4.01, josta on julkaistu myös XML-standardin toteuttava versio XHTML 1.0 [12]. Tuloillaan olevasta ja selaimien laajalti jo tukemasta HTML 5:stä on jo julkaistu kehitysversio [13].

2.3.2 CSS

CSS (Cascading Style Sheet, porrastetut tyyliarkit) on notaatio, jolla voidaan kuvata erilaisia tyyliominaisuuksia merkkauskielille. CSS:n yleisin käyttökohde on HTML-elementtien tyylien asettelu. CSS on suunniteltu, jotta dokumentin sisältö voidaan eriyttää dokumentin tyylistä. Eriytys voi parantaa sisällön saavutettavuutta ja vähentää tyylimääreiden toistoa, kun yhdellä tyylitiedostolla voidaan kuvata kokonaisen sivuston tyylit. CSS:n uusin versio on CSS 3, joka on oikeastaan kokoelma moduuleita, jotka laajentavat CSS 2.1 spesifikaatiota. [4]

2.3.3 JavaScript

JavaScript on protyyppipohjainen, tulkattava skriptikieli, joka on heikosti tyypitetty ja dynaaminen. Se on alunperin Netscapen kehittämä vuonna 1995 ja vuotta myöhemmin Ecma International hyväksyi sen standardoiduksi kieleksi nimellä ECMAScript [7]. Moderneissa selaimissa on JavaScript-tulkki, joka mahdollistaa websovelluksen toimintalogiikan ja HTML-tapahtumakäsittelijöiden suorittamisen käyttäjän selaimessa. Kielellä kirjoitetut komentosarjat voivat muokata HTML-sivun rakennetta käyttäen DOM-rajapintaa (Document Object Model) [41]. Myös tyyliä voi muokata komentosarjalla käyttäen hyödyksi HTML-elementtien tyyliominaisuutta (style-attribuutti). JavaScript-koodi kirjoitetaan yleensä erillisiin tiedostoihin, jotta komentosarjoista saadaan yleiskäyttöisempiä ja samalla ne pysyvät erillään HTML-merkkauksesta ja CSS-tyyleistä.

Eri selaimet sisältävät hieman erilaisia toteutuksia JavaScript-tulkista, mikä hankaloittaa web-sovelluksen tekemistä, koska komentosarjoja täytyy kirjoittaa erikseen eri selaimille. On kuitenkin olemassa kirjastoja, jotka on suunniteltu helpottamaan JavaScript-koodin kirjoittamista abstrahoimalla selainten eroavaisuudet. Yksi yleisimmin käytetyistä kirjastoista tähän tarkoitukseen on jQuery. jQuery helpottaa lisäksi DOM-rajapinnan käyttöä, tyylien muokkausta (CSS), tapahtumakäsittelijöiden toteutusta sekä Ajax-ohjelmointia. jQuery on lisäksi helposti laajennettavissa lukuisilla Internetistä löytyvillä laajennoksilla. [23]

2.3.4 Ajax-ohjelmointi

Ajax (Asynchronous JavaScript and XML) on ohjelmointityyli, joka yhdistää useita teknologioita. Ohjelmointityylin mahdollistaa XMLHttpRequest-rajapinta (lyhyesti XHR). XHR:n kehitti alunperin Microsoft Internet Explorer 5 -selainta varten ActiveX-objektina. Myöhemmin myös Mozilla-projektiin ja Applen Safari-selaimeen toteutettiin vastaava objekti. XHR:sta onkin muodostunut *de facto* -standardi ja lähes kaikki selaimet tukevat sitä. [5]

XHR mahdollistaa sisällön lataamisen palvelimelta käyttäjän selaimeen dynaamisesti, ilman, että koko selaimen näkymää tarvitsee päivittää. Kuten objektin nimi vihjaa, sisältö välitetään perinteisesti XML:nä, mutta käytännössä mikä tahansa merkkijono on mahdollinen välittää sisältönä. Ajax-ohjelmoinnilla voisi esimerkiksi toteuttaa puunäkymän, jossa solmun lapsisolmut ladattaisiin dynaamisesti kun solmu osoitetaan auki. Tällöin solmun osoitus laukaisisi tapahtumankäsittelijän, joka tekisi Ajax-kutsun palvelimelle, palvelin vastaisi esimeriksi palauttamalla lapsisolmut XHTML:nä, jonka jälkeen XHR-objektiksi paketoidusta vastauksesta voitaisiin ottaa sisältö ja sijoittaa se DOM-malliin sopivaan kohtaan.

2.3.5 JSON

JSON (JavaScript Object Notation) on kevyt tekstipohjainen avoin standardi olioiden esittämiseen. Se on osakokonaisuus ECMScriptin standardista tavasta luoda datarakenteita ja assosiatiivisia listoja [7]. JSON on kieliriippumaton ja kirjastot JSON-olioiden käsittelyyn on tarjolla lähes kaikkiin yleisimpiin ohjelmointikieliin. JSON-olio on mahdollista tulkata suoraan JavaScript-olioiksi, mikä on tehnyt siitä erityisen suositun Ajax-ohjelmoinnissa ja web-sovellusten tiedonsiirrossa. [17]

2.4 Testaus ja laadunhallinta

Jatkuva integraatio (Continuous Integration) ja automatisoitu testaus ovat ketterän ohjelmistokehityksen prosesseja, joilla parannetaan paitsi ohjelmiston laatua, myös vältetään monia integrointiin, julkaisuun ja ylläpitoon liittyviä ongelmia. Jatkuva integrointi tarkoittaa ohjelmiston koostamista ja osakokonaisuuksien integrointia jatkuvasti ohjelmistokehityksen ohella. Tässä kohdassa kuvataan sovelluksen testauksessa ja integroinnissa käytetyt tekniikat.

2.4.1 TestNG

Java-ohjelmistoalustalle on olemassa lukuisia erilaisia automatisoituun testaukseen tarkoitettuja ohjelmistokehyksiä. TestNG on yksi tällainen, joka on kehitetty kilpailevan suositun ohjelmistokehyksen JUnit:n inspiroimana. TestNG:n keskeisimmät edut ovat helppo integroiminen kehitysympäristöön ja käännösjärjestelmiin, sekä testien helppo ryhmittely ja ajojärjestyksen ja testien välisten riippuvuuksien määrittely. [39]

Automatisoidut testit ovat yleisimmin yksikkö- tai integrointitestejä, mutta myös koko järjestelmää testaavia testejä voidaan automatisoida. Yksikkötestit ovat matalan tason testejä, joilla ohjelmistokehittäjä varmistaa metoditasolla, että hänen kirjoittama ohjelmakoodinsa toimii tarkoitetulla tavalla. Integrointitestit ovat testejä,

joilla varmistetaan, että osakokonaisuuksien väliset rajapinnat toimivat suunnitellusti ja että kokonaisuudet hoitavat vastuualueensa. Järjestelmätestit taas testaavat koko järjestelmää sellaisena, kuin loppukäyttäjä sitä tulee käyttämään. Stripes-sovelluskehystä käytettäessä Java-koodia testaavilla testeillä voidaan kuitenkin testata vain palvelinsovellusta, eli tämän diplomityön aiheena toteutettavan järjestelmän järjestelmätestaaminen TestNG:llä ei ole mahdollista.

Testit voidaan suorittaa aina kun ohjelmakoodi käännetään, ohjelma suoritetaan tai jos testejä on hyvin paljon, aina ennen kuin muutokset ohjelmakoodiin viedään versionhallintaan muiden kehittäjien saataville. Yksikkötestauksella ei tietenkään voida saavuttaa virheetöntä järjestelmää, mutta ne ovat tehokas keino virheiden välttämiseksi ja auttavat erityisesti ketterässä ohjelmistokehityksessä, jossa uusia ominaisuuksia tehdään ja vanhoja muokataan jatkuvasti.

2.4.2 Selenium

Selenium on web-sovelluksen järjestelmätestaukseen suunniteltu sovelluskehys, joka automatisoidusti käyttää järjestelmää selaimen käyttöliittymän läpi. Testeillä on helppo testata todellisia käyttötilanteita, jolloin testatuksi tulee sekä asiakaspään käyttöliittymä, että palvelinsovellus. Yleisesti tietokonesovelluksien käyttöliittymää on hankala käyttää ohjelmallisesti, sillä käyttöliittymästä puuttuu komponenttien metatieto. Web-sovelluksessa käyttöliittymä on kuitenkin kuvattu HTML-merkkauksella, mikä mahdollistaa käyttöliittymäkomponenttien helpon tunnistamisen. Web-sovelluksen ohjelmallinen käyttö on mahdollista jopa käyttöliittymän muuttuessa, kunhan yksittäisten komponenttien tunnisteet eivät muutu. Kaikki yleisimmät selaimet ovat tuettuja, ja Selenium laatii raportin testien tuloksista jokaiselta ajolta. [34]

Selenium IDE on lisäosana selaimeen asennettava kehitysympäristö, jolla voidaan luoda Selenium-testitapauksia. Testitapaukset voidaan joko nauhoittaa käyttäen sovellusta manuaalisesti tai sitten ne voidaan kirjoittaa Seleniumin omaa DSL-merkkauskieltä (Domain Specific Language) hyödyntäen. [34]

2.4.3 Jenkins CI

Jenkins CI (Continuous Integration) on työkalu, jolla voidaan muun muassa automatisoida ohjelmiston kääntäminen ja paketointi aina, kun versionhallinnassa olevaan ohjelmakoodiin tehdään muutoksia. Työkalu sisältää kattavan valikoiman erilaisia lisäosia, joilla voidaan ajaa yksikkö-, itegraatio- ja järjestelmätestejä ohjelman kääntämisen yhteydessä. Ohjelmakoodia voidaan myös analysoida automatisoidusti erilaisin työkaluin, joilla voidaan esimerkiksi tunnistaa virheitä, heikkoa suunnittelua tai muita vastaavia ongelmia. [19]

Jenkins osaa lisäksi automaattisesti lähettää paketoidun sovelluksen sovelluspalvelimelle, joka voi automaattisesti pistää sen ajoon. Näin saadaan esimerkiksi testikäyttöä tai asiakasta varten aina ajantasainen ajoympäristö helposti pystytettyä. Jenkinsillä voi myös ajaa käyttöjärjestelmätason skriptejä, eli myös muiden rutiininomaisten ja aikaa vievien tehtävien automatisointi on mahdollista. [19]

3. PROJEKTIN TAUSTA

Tässä luvussa kuvataan projektin taustat, eli projektissa toteutettavan järjestelmän tilannut asiakas ja asiakkaan liiketoimintaan liittyvät muut järjestelmät. Tutustumme tilatun järjestelmän toimintaympäristöön ja kuinka se liittyy asiakkaan palvelukokonaisuuteen. Lisäksi käydään läpi järjestelmälle asetetut vaatimukset, suunnitteluperiaatteet ja vaatimukset täyttävän ohjelmiston kuvaus.

3.1 Järjestelmän toimintaympäristö

Asiakas, jolle musiikin tilaus- ja hallintajärjestelmä toimitetaan, on Pohjoismaiden suurin taustamusiikkipalveluita tarjoava yritys. Yrityksen ydinosaamistaan on taustamusiikkipalveluiden tarjoaminen esimerkiksi hotelleille, ravintoloille, yökerhoille, kauppakeskuksille ja muille vastaaville toimijoille. Yritys on perustettu 1990-luvulla ja asiakkaan toisto-ohjelma on ollut Suomen ensimmäinen tietokonepohjainen taustamusiikkisoitin.

Asiakkaan palvelukokonaisuuksien tai niiden osa-alueiden loppukäyttäjiä eli asiakkaan asiakkaita kutsutaan tässä työssä käyttäjiksi. Järjestelmän toimittajalla tarkoitetaan siis käyttäjien näkökulmasta palveluntarjoajaa, joka on tämän diplomityön ja projektin asiakas.

3.1.1 Yleiskuvaus ja liittyvät järjestelmät

Asiakkaan palveluita käytetään jokseenkin yhtenevissä ympäristöissä tai ainakin niihin kaikkiin liittyy yhteinen taustamusiikin tarve. Järjestelmien loppukäyttäjät eroavat kuitenkin suuresti eri kohteiden välillä. Esimerkiksi yökerhossa tuotteita saattaa käyttää vuosien kokemuksen musiikkiohjelmien käytöstä omaava huippu-dj, kun taas hotellissa taustamusiikkisoitinta saattaa käyttää hotellin henkilökunta tai ravintolassa tarjoilija.

Musiikkialan ammattilaisilla on hyvin erilaiset tarpeet ja odotukset järjestelmän suhteen kuin alaa ja tekniikkaa tuntemattomalla henkilöllä. Yhteistä käyttäjille kuitenkin on se, että heidän tulisi järjestelmien avulla pystyä saavuttamaan tavoitteensa turhautumatta. Tämä asettaa toimitettavalle web-palvelulle tiukat rajoitteet ja odotukset niin asiakkaan kuin loppukäyttäjienkin taholta. Johdonmukaisuuden asiakkaan muiden järjestelmien kanssa tulisi olla hyvä, ja toisaalta palvelun tulisi

olla johdonmukainen myös toiminnoiltaan kutakuinkin vastaavien kuluttajille suunnattujen palveluiden kanssa.

Kuvassa 3.1 on esitetty asiakkaan palvelukokonaisuuden muodostavat järjestelmät ja niiden muodostama palvelukokonaisuus. Kuvasta löytyvät seuraavat osajärjestelmät:

- **Toisto-ohjelma**: Käyttäjän toimipisteeseen päätelaitteelle asennettava sovellus, jolla toistetaan päätelaitteelle ennalta ladattua musiikkia, musiikkivideoita, karaokevideoita ja mainoksia. Toisto-ohjelma voidaan automatisoida soittamaan itsenäisesti soittolistoja tai musiikkiprofiileita.

- **Latausohjelma**: Käyttäjän päätelaitteelle asennettava taustasovellus, joka päivittää päätelaitteen musiikkikirjastoa vastaamaan palvelimella ylläpidettyä käyttäjän mediakirjaston sisältöä. Lataa mediatiedostot palvelimelta käyttäjän päätelaitteelle käyttäen toimitilan Internet-yhteyttä.

- **Palvelinohjelmisto**: Palveluntarjoajan palvelimelle asennetut sovellukset, jotka tarjoavat rajapinnat käyttäjän päälaitteelle asennettujen sovellusten käyttöön. Tarjoavat tietoa käyttäjästä, käyttäjän mediakirjastosta ja muista asiakkuuteen liittyvistä asioista. Toimivat myös mediatiedostojen tarjoajana latausohjelmalle.

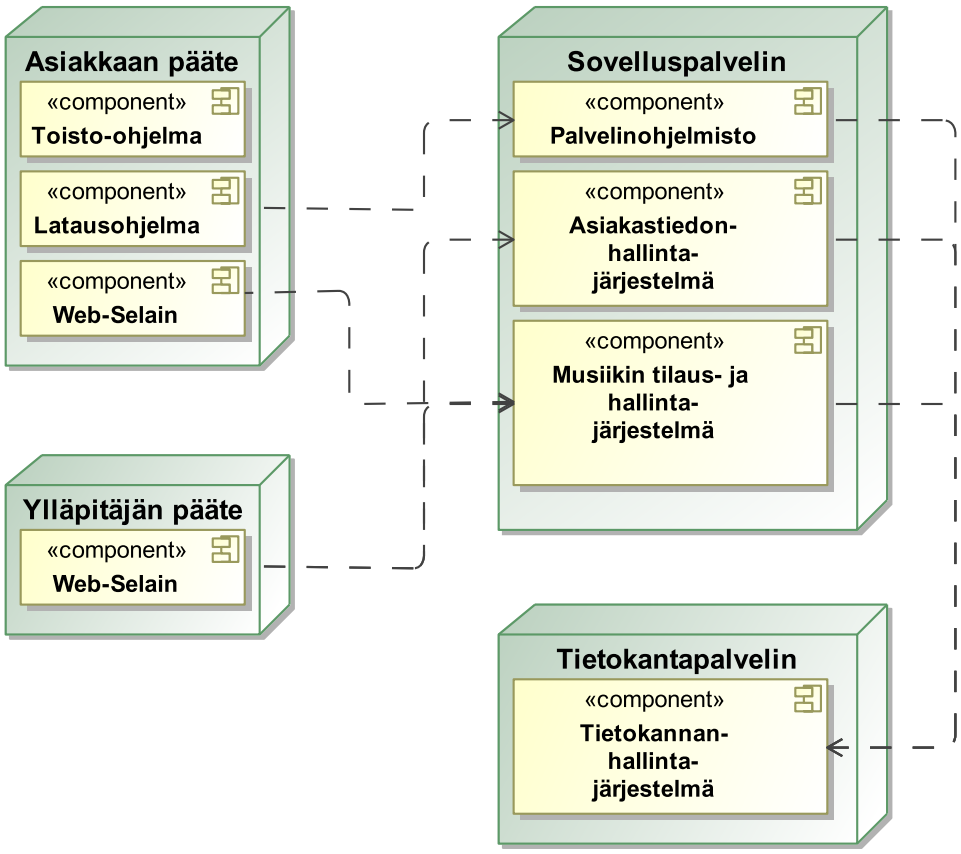
- **Tietokannanhallintajärjestelmä**: Hallitsee osajärjestelmien kesken yhteisesti jaettua tietokantaa.

- **Asiakastiedon hallintajärjestelmä**: Web-pohjainen järjestelmä, jonka avulla voidaan hallita ja tarkastella asiakastietoja, tilaustietoja, tilastoja ja hallita mediavalikoimaa. Tämä järjestelmä on vain palveluntarjoajan sisäisessä käytössä.

- **Musiikin tilaus- ja hallintajärjestelmä**: Järjestelmä, jonka avulla loppukäyttäjät voivat hallita asiakkuustietojaan, mediakirjastojensa sisältöä, soittolistoja, soittoprofiileita, mainoksia ja lähettävää viestejä ylläpidolle. Järjestelmän ensimmäinen suppeampi versio on toteutettu aikaisemmin asiakkaan toimesta ja tämän diplomityön aiheena on järjestelmän toinen, kaikki ominaisuudet kattava versio.

3.1.2 Erilaiset palvelukokonaisuudet

Kaikki käyttäjät eivät tarvitse kaikkia osajärjestelmiä, ja palvelut onkin jaettu erilaisiin kokonaisuuksiin. Suppein paketti sisältää pelkästään web-pohjaisen musiikin



Kuva 3.1: Palvelukokonaisuus

tilaus- ja hallintajärjestelmän, sekä käyttöliittymättömän version taustamusiikkisoittimesta. Tämän paketin käyttäjät tilaavat musiikin, luovat soittolistoja ja ajastavat soittolistat musiikin tilaus- ja hallintajärjestelmässä, jonka jälkeen soitin automaattisesti toistaa taustamusiikin käyttäjän tiloissa.

Vaativampiin käyttökohteisiin on tarjolla paketti, johon sisältyy edellisten toimintojen lisäksi käyttöliittymällinen taustamusiikkisoitin ja laajemmat ominaisuudet web-palveluun. Käyttäjällä voi lisäksi olla useita eri kohteita, joissa voi olla erilaiset mediakirjastot ja jokainen kohde voi lisäksi koostua useista eri tiloista. Kussakin tilassa tulee voida soittaa eri musiikkia. Tehokäyttäjille on olemassa palvelupaketti, joka kattaa kaikki tarjolla olevat palvelut ilman rajoituksia.

3.1.3 Tiedot ja tietokanta

Tässä alakohdassa kuvataan järjestelmän tietosisältö ja tietojen pysyvyydestä ja saatavuudesta vastaava tietokanta. Kohdassa kuvataan myös tietokannan arvioitu käyttöintensiteetti ja tietokannanhallintajärjestelmän kapasiteettivaatimukset.

Tietosisältö

Osajärjestelmät integroituvat kokonaisuuteen tietokannan avulla. Tietokannasta vastaa tietokannanhallintajärjestelmä, joka on asiakkaan valitsema ja käyttöönottama MySQL. Se on maailman käytetyin tietokannanhallintajärjestelmä ja sen omistaa Oracle Corporation. MySQL on saatavilla kahdella eri lisenssillä, joista toinen on vapaa (GPL-lisenssi [10]) ja toinen kaupallinen. [27]

Tietokanta sisältää metatiedon kaikesta asiakkaan mediavalikoimasta, eli palvelimelta löytyvät musiikkikappaleet, musiikkivideot, karaokevideot ja -paketit sekä mainokset. Kanta sisältää myös tilastotiedot käyttäjien mediatoistosta, käyttäjien ja ylläpidon väliset viestit sekä käyttäjien soittolistat ja soittoprofiilit. Tilastotiedot tallennetaan kaikista soitetuista kappaleista, katsotuista musiikkivideoista, tilauksista ja kaikesta muusta minkä tilastoimisesta voidaan tarvittaessa saada lisäarvoa.

Kuvassa 3.2 on esitetty tietokannan pääkäsitteet ja niiden väliset suhteet. Tietomallissa loppukäyttäjät liittyvät aina yhteen järjestelmän asiakkaaseen. Kaikki tilaukset ja ostoskori ovat siis asiakaskohtaisia, eivät käyttäjäkohtaisia. Asiakkaaseen liittyy myös järjestelmän asetukset, jotka määrittävät esimerkiksi mitä toimintoja käyttäjillä on käytettävissä.

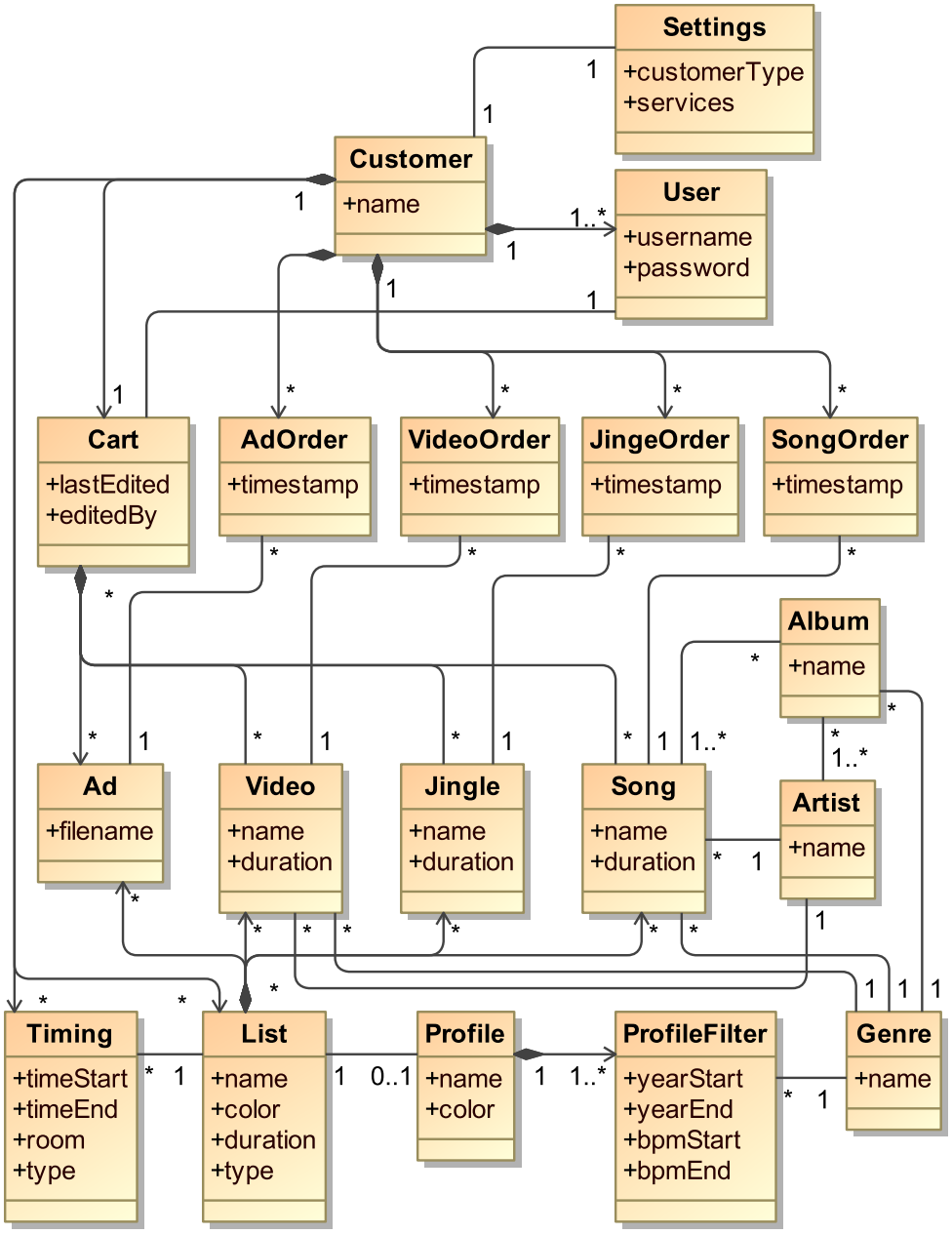
Käyttöintensiteetti

Nykyisellään järjestelmällä on käyttäjiä muutamia tuhansia ja käyttäjien määrä on kasvusuhdanteinen. Tietokannan käyttö on jatkuvaa, sillä monissa käyttökohteissa taustamusiikkia toistetaan läpi viikon aukioloaikojan puitteissa. Kohteiden vaihdellessa kauppakeskuksista pubeihin, tarkoittaa tämä lähes ympärivuorokautista käyttöä. Intensiivisintä käyttö on kuitenkin viikonloppuisin ja erityisesti perjantai- ja lauantai-iltaisin, jolloin suurin osa käyttäjistä paitsi toistaa musiikkia myös hallitsee mediakirjastojaan.

Kapasiteettivaatimukset

Käytännössä tietokannanhallintajärjestelmän tulee pystyä palvelemaan kaikkia käyttäjiä yhtä aikaa tilastotietojen keräämisen puitteissa ja metatiedon hakujakin tulisi pystyä tarjoamaan yhtäaikaisesti ainakin puolelle käyttäjäkunnasta. Tietokannalle raskaimpia operaatioita ovat erilaiset haut mediavalikoimiin, sillä mediavalikoimat ovat laajoja ja haut koskevat usein laajaa osaa kannan sisällöstä.

Tietokannan kapasiteettivaatimukset ovat tämän projektin vastuualueen ulkopuolella, mutta ne on hyvä tiedostaa, sillä web-palvelun tiedonhallinnalla voi olla merkittävä osuus projektin onnistumisessa. Tämä tulee ottaa huomioon erityisesti web-palvelun hakuja suunniteltaessa, sillä jos haku saadaan rajattua vain osaan tietosisällöstä, saadaan raskaita hakuja kevennettyä ja kapasiteettivaatimuksia pienen-



Kuva 3.2: Tietomalli

nettyä. Toisaalta osa käyttäjien tiedoista voidaan hallita paikallisesti web-palvelun tietovarastokerroksella, jolloin toistuvat kyselyt saadaan kokonaan karsittua pois tietokannanhallintajärjestelmän työkuormasta.

3.2 Musiikin tilaus- ja hallintajärjestelmä

Tässä kohdassa kuvataan miksi asiakas tarvitsee järjestelmän ja millaisia ongelmia järjestelmän tulee pystyä ratkaisemaan. Käymme läpi myös järjestelmän ensimmäisen version sisältämät toiminnot, miksi järjestelmä tehtiin ja miksi tarvitaan uusi

versio järjestelmästä.

3.2.1 Tarve

Taustamusiikkipalvelut ovat yhä kasvava ala, eivätkä asiakkaan ydinliiketoiminta tai sen edellytykset ole muuttuneet. Maailma on kuitenkin hyvin erilainen kuin 1990-luvulla, ja erityisesti Internet ja palvelukeskeinen ajattelu luovat painetta palveluiden kehittämiseen. Musiikkiala ja musiikin kuluttaminen ovat kokenut suuren muutoksen Internet-aikana, jonka seurauksena kuluttajien tottumukset ja odotukset musiikkipalveluiden suhteen ovat muuttuneet myös.

Musiikkikirjaston sisällön hallinta ja organisointi ovat irrallisia toimintoja musiikin soittamisesta ja mahdollisesti myös muusta käyttäjän tiloissaan tekemästä liiketoiminnasta. Usealla käyttäjällä on lisäksi toimipaikalla ollessaan kädet täynnä muita töitä ja aikaa esimerkiksi soittolistojen miettimiselle ei ole. Onkin loogista, että halutessaan käyttäjän tulisi voida tehdä musiikkikirjaston hallintaa muualtakin, kuin taustamusiikkia soittavalta päätteeltä.

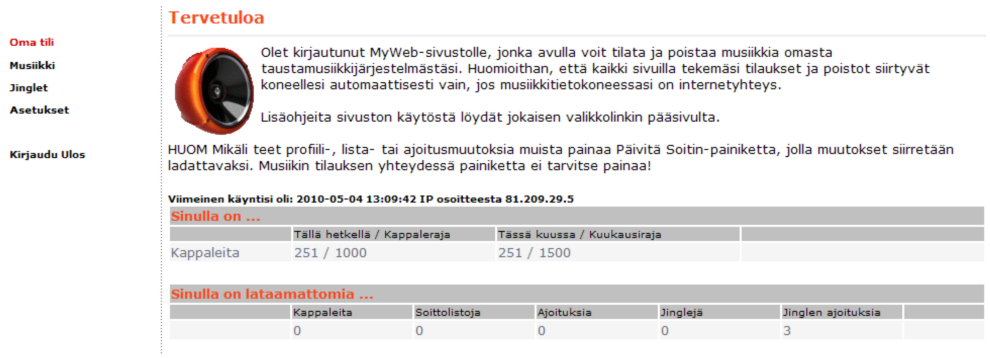
Vastatakseen uudenlaiseen kysyntään ja muuttuvaan toimintaympäristöön asiakas on kehittänyt taustamusiikkipalvelustaan web-pohjaisen version, jossa käyttäjät hallitsevat toimitilansa taustamusiikkia pelkän web-käyttöliittymän avulla. Järjestelmän Internet-pohjaisuus tarjoaa riippumattomuuden sijainnista, kellonajasta ja tilanteesta, kunhan Internet-yhteys on saatavilla. Jatkuvasti päivittyvä sisältö, uudet ideat ja sovelluspäivitykset ovat lisäksi vaivaton suunnitella ja toteuttaa keskitetylle palvelimelle, kun käyttäjien päätteille ei tarvitse tehdä päivityksiä tai muutoksia.

3.2.2 Ensimmäinen versio

Asiakas toteutti järjestelmän ensimmäisen version itse. Lähtökohtana järjestelmän toteutukselle oli nopea käyttöönotto ja näin nopeasti käyttäjiltä saatava palaute. Laajennettavuus ja skaalautuvuus eivät olleet järjestelmän vaatimuksia, eikä järjestelmää näin ollen hyödynnetty toisen version toteutuksen pohjana. Kuvassa 3.3 on kuvakaappaus järjestelmän ensimmäisen version käyttöliittymästä.

Tilaus- ja hallintajärjestelmän ensimmäinen version tukemia toimintoja ovat:

- **Musiikin hakeminen**: Käyttäjät pystyivät tekemään hakuja tietokantaan ja järjestelemään hakutuloksia.
- **Musiikkikirjaston hallinta**: Muusiikkikappaleita pystyi tilaamaan ja poistamaan tilauskorin avulla.
- **Äänimainoksien hallinta**: Äänimainoksia (jinglet) pystyi tilaamaan, poistamaan ja ajastamaan.



Kuva 3.3: Järjestelmän ensimmäinen versio

- **Soittolistojen hallinta**: Soittolistoja pystyi lisäämään ja poistamaan, sekä soittolistojen sisällön pystyi valitsemaan. Soittolistojen kappaleita ei kuitenkaan voinut järjestää.
- **Soittoprofiilien hallinta**: Soittoprofiileita pystyi luomaan ja poistamaan oman musiikkikirjaston sisällöstä.
- **Asiakkuustietojen tarkastelu**: Asiakkuuteen liittyviä tietoja pystyi tarkastelemaan, mutta ei muokkaamaan.

3.3 Kohti seuraavaa versiota

Järjestelmän ensimmäinen otettiin hyvin vastaan ja asiakas päättikin lähteä laajentamaan palvelua. Uusia tarvittavia toimintoja olivat muun muassa viestien lähetys käyttäjien ja järjestelmän toimittajan välillä, musiikkivideoiden, karaokevideoiden ja mainoksien esikatselu ja tilaus, monipuolisempi ja helpompi soittolistojen laatiminen, helpompi ja intuitiivisempi ajastuksien teko ja kolmansien osapuolten ylläpitämät eräänlaiset dynaamiset soittolistat. Ohjelmistoja tuottava asiakas ei kuitenkaan lähtenyt toteuttamaan uutta laajempaa järjestelmää itse, sillä aihealue vaati hieman laajempaa osaamista web-järjestelmistä ja toisaalta asiakkaalla ei ollut riittäviä resursseja toteuttaa projektia halutussa aikataulussa.

Asiakkaalla on valtava määrä arvokasta tietoa Pohjoismaiden suosikkimusiikista tietokannassaan. Yksi projektin tavoitteista onkin, että tieto saadaan hyödynnettyä ja järjestelmän käyttäjille annetaan mahdollisuus hyötyä Internetin yhteisöllisyydestä. Käyttäjien tulisi voida jakaa soittolistojaan ja nähdä mitkä kappaleet ovat kuunnelluimpia ja ladatuimpia.

Tässä kohdassa kuvataan järjestelmän uuteen versioon suunnitellut näkymät ja toiminnot.

3.3.1 Yleiskuvaus

Musiikin tilaus- ja hallintajärjestelmän toinen versio on palveluntarjoajan asiakkaille tarkoitettu ekstranet-palvelu. Palvelun tyylisuunnaksi on valittu *vähemmän tekstiä ja enemmän ilmettä*. Järjestelmää tullaan käyttämään myös kosketusnäytöillä, joten painikkeiden ja kontrollien tulee olla riittävän suuria, jotta niitä voi sujuvasti käyttää myös ilman osoitinta.

Palvelussa käyttäjän on helppo nähdä mikä sisältö on uutta ja suosittua, ja käyttäjän on vaivatonta tilata sisältö omaan järjestelmäänsä. Näkymät ja käyttöliittymä yksilöidään asiakkaan palvelukokonaisuuden mukaan, eli asiakkuuden tyyppiin kuulumattomat ominaisuudet poistetaan dynaamisesti käytöstä. Ominaisuuksia, kuten karaoke, musiikkivideot ja mainosvideot ei kuitenkaan tule poistaa kokonaan näkyvistä, vaan ne pitää jäädä käyttöliittymään näkyviin harmaina, jotta käyttäjät huomaavat ominaisuuksien olemassaolon ja mahdollisesti haluavat myöhemmin päivittää palvelukokonaisuutensa kattamaan kyseiset ominaisuudet.

Palvelun tulee kattaa kaikki ensimmäisen version sisältämät ominaisuudet, mutta ominaisuudet tehdään käytettäviksi uusien intuitiivisempien ja käytettävämpien käyttöliittymien avulla. Tilaukset ja mediakirjaston hallinta tehdään verkkokauppamaiseen tyyliin tilauskoria hyödyntäen, johon käyttäjät voivat kerätä tilattavaa ja poistettavaa sisältöä. Tilaukset menevät aina ensin tilauskoriin, jonka hyväksymällä uusi sisältö siirtyy asiakkaan mediakirjastoon. Taustajärjestelmä huolehtii uuden sisällön lataamisesta asiakkaan järjestelmään, kun tilauskori on hyväksytty.

Ajastuksia ja soittolistoja tulee voida tehdä musiikin lisäksi musiikkivideoista, mainosvideoista ja kuvamainoksista. Ajastukset voidaan tehdä tilakohtaisesti ja niitä tulee voida muokata ja tarkastella viikkokalenterimaisessa näkymässä. Käyttäjien tulee pystyä tarkastelemaan ja muokkaamaan asiakastietojaan, vaihtamaan salasanansa ja tilaamaan lisää palveluita lähettämällä viestejä myynnille. Viestejä tulee pystyä lähettämään erilaisiin palveluntarjoajan postilaatikkoihin ja palveluntarjoajan tulee pystyä lähettämään viestejä yksittäisille käyttäjille tai käyttäjäryhmille.

Lisäksi käyttäjien tulee voida ladata palveluun omia kuvamainoksiaan, äänimainoksiaan ja videomainoksiaan. Omia mainoksia voi lisätä mainossoittolistoille ja ajastaa samaan tapaan kuin muutakin sisältöä.

3.3.2 Näkymille yhteiset toiminnot

Kuvassa 3.4 on hahmotelma järjestelmän yleisilmeestä. Sivun yläosassa on kiinteästi esillä logo, murupolku, kaksitasoinen navigointi osio, pikahaku sekä linkki tilauskoriin. Yläosassa tulisi lisäksi olla näyteikkunamainen interaktiivinen palkki, jolla käyttäjä voi nopeasti selata uusimpia albumeita. Albumia osoittamalla tulisi aueta käyttöliittymän päälle ikkuna, jossa on listattuna albumin kappaleet ja napit joista

albumin tai yksittäisiä kappaleita voi tilata. Sivun keskiosa on varsinaista sisältöä varten ja sivun alalaidassa on sivukartta.

Kuva 3.4: Käyttöliittymähahmotelma

Mediasisältöä tulee voida esittää ainakin kolmella eri tavalla. Ruudukkoesitys (kuvassa 3.4) on visuaalisesti houkutteleva esitystapa, jossa esimerkiksi musiikkialbumeiden kannet on aseteltuna ruudukkoon. Albumien kuvien alla näytetään albumin ja artistin nimet.

Toinen esitystapa on listaesitys, joka on yksinkertainen ja tiivis taulukko, josta selviää paljon tietoa yhdellä silmäyksellä. Listaa käytetään esimerkiksi hakujen tuloksien esittämiseen tai jos esitettävää dataa on muuten liian paljon yhdellä sivulla esitettäväksi. Listan voi järjestää sarakkeiden otsikoita osoittamalla yhden tai useamman sarakkeen mukaan. Listan tulee myös tukea sivutusta ja yhdellä sivulla näytettävien rivien lukumäärän vaihtamista.

Kolmas esitystapa on yhdistelmä listoja ja kuvia, jolla voidaan esittää esimerkiksi

artistin albumit. Albumiesityksessä näytetään jokaisen albumin kansikuva isona ja kannen oikealla puolella näkyy artistin nimi, albumin julkaisuvuosi sekä lista kappaleista, jotka albumilta löytyvät. Albumit ovat albumiesityksessä allekkain väliviivalla eroteltuina.

Jokaisesta näkymästä tulee pystyä lisäämään sisältöä tilauskoriin ja toisaalta tulee voida nähdä mitkä kappaleet tai videot on jo tilauskorissa. Lisäksi musiikkikappaleita tulee pystyä esikuuntelemaan ja videoita tulee voida esikatsella viemällä kursori videon kuvan päälle.

3.3.3 Asiakkuuden hallintaan liittyvät näkymät

Päätason navigointipalkin ensimmäisen välilehden alta löytyvät linkit asiakkuuden hallintaan liittyville sivuille. Oma tili -sivulle pääsee alemman tason navigointipalkin ensimmäistä linkkiä osoittamalla. Tällä sivulla esitetään käyttäjän asiakastiedot, sopimustiedot sekä statistiikkoja käyttäjän mediakirjastosta ja sopimukseen liittyvistä saldoista. Toinen linkki vie Omat tiedot -näkymään, jossa käyttäjät voivat muokata asiakastietojaan, vaihtaa palvelun kielen sekä vaihtaa salasanansa.

Viestit sivulta käyttäjät voivat käydä lukemassa heille lähetetyt ja heidän lähettämät viestit. Tältä sivulta voi lisäksi lähettää uuden viestin tai kappaletoiveen. Viestien lukeminen ja kirjoitus tulee tapahtua käyttöliittymän päälle avautuvassa erillisessä ikkunassa.

3.3.4 Mediakirjastoon ja toistoon liittyvät näkymät

Päätason toiselta välilehdeltä löytyvät käyttäjän mediakirjastoon ja median toistoon liittyvät näkymät. Oma soitin -sivulla käyttäjä voi tarkastella mediakirjastonsa yksityiskohtaisia statistiikkoja ja sopimukseen liittyviä mahdollisia saldorajoja. Media-sivulla esitetään kaikki käyttäjän mediakirjaston sisältö. Esitystapa on käyttäjän valittavissa ja käyttäjä voi tehdä mediakirjastoon erilaisia hakuja ja rajata sekä järjestää hakutuloksia erilaisten ehtojen perusteella.

Profiilit-sivulla käyttäjä voi luoda, muokata ja poistaa soittoprofiileita. Profiilien muokkausnäkymässä tulee olla päivittyvä lista profiiliin sopivista kappaleista, ja käyttäjä voi valita luodaanko lista käyttäjän mediakirjastosta löytyvistä kappaleista vai kaikesta palvelusta löytyvästä musiikista.

Soittohistoria-sivulla käyttäjä voi tehdä hakuja soittohistoriaansa. Haun tuloksia tulee voida rajata päivämäärien, kellonaikojen ja toistetun median tyypin perusteella.

Soittolistat-sivulla käyttäjä voi luoda, muokata ja poistaa soittolistoja. Soittolistat voivat sisältää musiikkia, mainoksia tai musiikkivideoita. Listat luodaan näkymässä, jossa vasemmalla on käyttäjän mediakirjaston sisältö, sekä hakukenttä jolla

palveluun voidaan tehdä hakuja. Oikeassa laidassa on itse soittolista kappaleiden soittojärjestyksessä. Mediaa voidaan vetää soittolistalle vasemman laidan taulukoista, eli mediakirjastosta tai kaikesta palvelun sisällöstä hakua käyttäen. Soittolistaa tulee voida myös järjestää vetämällä kappaleita ylös- tai alaspäin listalla.

Ajastukset-sivulla näytetään kalenteria muistuttava viikkonäkymä, jossa kutakin viikonpäivää edustaa yksi sarake ja ajastukset näkyvät laatikkoina sarakkeissa. Ajastuksien kellonaikaa, päivää ja kestoa voidaan vaihtaa vetämällä tai venyttämällä laatikkoa. Näkymässä näkyviä ajastuksia voidaan lisäksi suodattaa tyypin ja ajastuksen kohteena olevien tilojen avulla. Käyttäjä voi esimeriksi suodattaa näkymään vain aulabaarin musiikkiajastukset.

3.3.5 Musiikkinäkymät

Musiikkiin liittyvät näkymät löytyvät navigoinnin päätason kolmannelta välilehdeltä. Musiikkia listaavia sivuja on useita: esimerkiksi uusi musiikki, soitetuin musiikki ja ladatuin musiikki. Käyttäjän tulee voida valita esitystapa kullakin musiikkisivulla erikseen.

Musakorit ovat palveluntarjoajan laatimia listoja musiikkikappaleista, jotka sopivat hyvin yhteen tai ovat muuten hyvä kokonaisuus. Musakorit-sivulla käyttäjät voivat selata ennalta laadittuja koreja, tarkastella niitä ja tilata korissa olevia yksittäisiä kappaleita tai halutessaan koko musakorin.

Live-listat ovat listoja, jotka päivittyvät tilastotietojen perusteella, palveluntarjoajan manuaalisesti päivittäminä tai esimerkiksi yhteistyökumppanin toimesta. Ne tarjoavat käyttäjille helpon tavan tilata automaattisesti päivittyviä listoja, jolloin musiikkia ei tarvitse aktiivisesti itse etsiä. Live-listat esitetään ja tilataan samaan tapaan kuin musakorit.

3.3.6 Muut näkymät

Päätason navigointipalkista löytyvät myös välilehdet musiikkivideoita, karaokea ja mainoksia koskevia sivuja varten. Karaoke-sivulla käyttäjät voivat selata erilaisten karaokepakettien sisältöä valitsemallaan esitystavalla ja esikatsella karaokevideoita. Karaokepakettien tilaus tapahtuu ottamalla yhteyttä palveluntarjoajan myyntiin esimerkiksi lähettämällä viestin palvelussa.

Mainokset-sivulta käyttäjät voivat ladata palveluun omia kuva-, ääni- ja videomainoksiaan. Käyttäjät voivat myös tilata ja esikuunnella palvelusta löytyviä äänimainoksia.

Käyttäjät voivat tilata musiikkivideoita yksittäin kuten musiikkikappaleitakin. Videoiden selailu tapahtuu ruudukko- tai listanäkymässä ja myös musiikkivideoita voi esikatsella.

Jokaisella mediatyypille tulee olla oma tehokasta ja tarkkaa hakua varten oleva hakusivu. Lisäksi palvelussa tulee olla pikahaku, jonka hakukenttä on sivun ylälaidassa. Pikahaku hakee kaikesta mediasisällöstä ja esittää hakutulokset mediatyypeittäin luokitelluissa taulukoissa.

Tilauskorissa, johon pääsee sivujen ylälaidasta löytyvällä painikkeella, on kaksi osiota: tilaukset ja poistot. Kummassakin osiossa sisältö on lisäksi luokiteltu mediatyypeittäin ja jokaisen osion voi piilottaa tai näyttää erikseen. Tilauskorin voi tyhjentää yhdestä painikkeesta ja hyväksyä toisesta. Jokaisen tilausrivin voi lisäksi poistaa erikseen.

Näkymien suunniteltu asettelu ja sisältö on suuntaa-antava ja ne tuleekin toteuttaa niin, että sisältöä on helppo organisoida uudelleen tarvittaessa. Sisäinen johdonmukaisuus on lisäksi tärkeässä asemassa, koska monet sivuista poikkeavat toisistaan vain sisällön ja mediatyypin perusteella.

3.4 Yleiset vaatimukset

Toteutustekniikoiden suhteen asiakkaalla ei ollut suuria vaatimuksia järjestelmälle. Yhdessä asiakkaan kanssa sovittiinkin, että järjestelmä toteutetaan Java-ohjelmistoalustalle käyttäen Stripes-sovelluskehystä. Integroituminen asiakkaan muihin järjestelmiin tuli tapahtua kokonaisuuden tavoin käyttäen järjestelmien kesken jaettua tietokannanhallintajärjestelmää.

Toteutetun järjestelmän haluttiin lisäksi olevan seuraavien ohjelmistoille yleisten vaatimusten mukainen:

- **Avoin**: Järjestelmän tulee käyttää ja tukea yleisiä protokollia sekä standardeja (HTTP, XML, HTML). Avoimuudella vältetään yhteensopivuusongelmat nyt ja tulevaisuudessa.

- **Laajennettava**: Uusien ominaisuuksien toteutus tulee onnistua niin, ettei olemassa olevaan ohjelmakoodiin tarvitse tehdä mittavia muutoksia. Toisaalta olemassa olevan ohjelmakoodin ja toimintojen tulee olla mahdollisimman helposti muokattavissa. Oletettavaa on, että järjestelmää tullaan laajentamaan lähitulevaisuudessa.

- **Luotettava**: Järjestelmä ei saa olla altis virheille. Mikäli virheitä havaitaan, tulee niistä toipua hallitusti ja virheet sekä niihin johtaneet käyttäjän toimet tallentaa lokiin.

- **Moderni**: Toteutustekniikoiden tulee olla ajanmukaisia, jotta järjestelmä palvelee mahdollisimman pitkään. Toisaalta valistuneimmankin käyttäjän tulee saada järjestelmästä edustava ja teknisesti houkutteleva kuva.

- **Riippumaton**: Järjestelmän tulee olla riippumaton kolmansista osapuolista, eli jos toteutuksessa käyttää kolmannen osapuolen kirjastoa, tulee kirjaston käyttö ja päivitys olla kontrolloitua. Toteutuksessa ei saa myöskään käyttää sellaisia kirjastoja, joiden lisenssit ovat maksullisia.

- **Skaalautuva**: Palvelun tulee pystyä vastaamaan asiakkaan kasvutavoitteiden mukaisiin käyttäjämääriin myös vuosien päästä.

- **Suorituskykyinen**: Järjestelmällä ei tule olla kohtuuttoman kovia suorituskykyvaatimuksia, jotta sen ylläpito on edullista ja laitteistovaatimukset ovat linjassa asiakkaan muiden järjestelmien kanssa.

- **Tietoturvallinen**: Tietoliikenteen tulisi olla salattua ja palveluun pääsy tulisi sallia vain asiakkaan tietokantaan rekisteröidyille käyttäjille. Myös mahdollisiin hyökkäyksiin tulee varautua välttämällä tunnetut haavoittuvuudet, kuten esimerkiksi XSS (Cross-Site Scripting) ja DoS (Denial of Service) [8, 26].

4. JÄRJESTELMÄN TOTEUTUS

Ohjelmistoarkkitehtuuri tarkoittaa tapaa, jolla ohjelmiston eri osat, osien keskinäiset suhteet ja suhteet ympäristöön on organisoitu keskenään, sekä periaatteita, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota [14]. Ohjelmiston suunnittelun tarkoituksena on muuntaa ohjelmiston määrittely tekniselle kielelle, eli järjestelmän toteutuksen kuvaukseksi [11]. Tässä luvussa kuvataan toteutetun järjestelmän arkkitehtuuri pääperiaatteineen, ohjelmiston suunnittelu, toteutus ja testaus, sekä joitain käyttöönottoon ja ylläpitoon liittyviä seikkoja.

4.1 Yleisarkkitehtuuri

Toteutustekniikat ja arkkitehtuuri ovat riippuvaisia toisistaan, ja usein ohjelmistoalustan käyttö ja laajan sovelluskehyksen valinta, ohjaavat järjestelmän arkkitehtuuria johonkin suuntaan. Tämä toisaalta helpottaa järjestelmän suunnittelua, mutta vaarana on, että huonot teknologiavalinnat johtavat heikkoon arkkitehtuuriin. Järjestelmän tekniikat valittiin aiemmista järjestelmätoteutuksista saatujen kokemuksien perusteella ja niiden tiedettiin soveltuvan hyvin tällaiseen järjestelmään. Arkkitehtuuriin vaikuttavia tekniikoita tässä työssä ovat Java-ohjelmistoalusta ja Stripes-sovelluskehys, jotka valittiin yleisarkkitehtuurin suunnittelun yhteydessä.

4.1.1 Suunnitteluperiaatteet

Yleisarkkitehtuuriin vaikuttavia asiakasvaatimuksia ovat:

- Muiden asiakkaan järjestelmien kanssa jaettu tietokanta luo lähtökohdan ja tiukan rajoitteen järjestelmän korkeantason arkkitehtuurille. Jotta tietoliikenneyhteys asiakkaan tietokantapalvelimeen ei muodostu suorituskyvyn pullonkaulaksi, on järjestelmän oltava asennettavissa asiakkaan palvelintilaan. Toisaalta sovelluksen tietorakenne on myös jo tiedossa, mikä osittain helpottaa, mutta toisaalta rajoittaa arkkitehtuuripäätöksiä.

- Integroituminen muihin järjestelmiin tietokannan avulla luo rajoitteita välimuistien käyttöön. Tieto saattaa päivittyä kantaan toisen järjestelmän toimesta, jolloin välimuistiin tallennettu tieto on vanhentunutta.

- Suorituskyvyn tulee olla hyvä ja tietokantakyselyiden lukumäärä ja niiden raskaus tulee saada minimoitua ilman, että tietojen oikeellisuus ja saavutettavuus

kärsii. Tietokannan indeksointi on myös ongelma, sillä sitä ei voi suunnitella yhden sovelluksen näkökulmasta, koska tietokanta on jaettu usean sovelluksen kesken.

- Navigointipalkkeihin ja tiedon sivutukseen perustuva web-palvelun asettelu luo myös rajoituksia arkkitehtuurin kannalta. Yleiset sivuille yhteiset komponentit tulee tunnistaa ja navigointi, sekä sivuston lataus suunnitella niin, että sivut latautuvat nopeasti ja sisältävät sopivan määrän sisältöä.

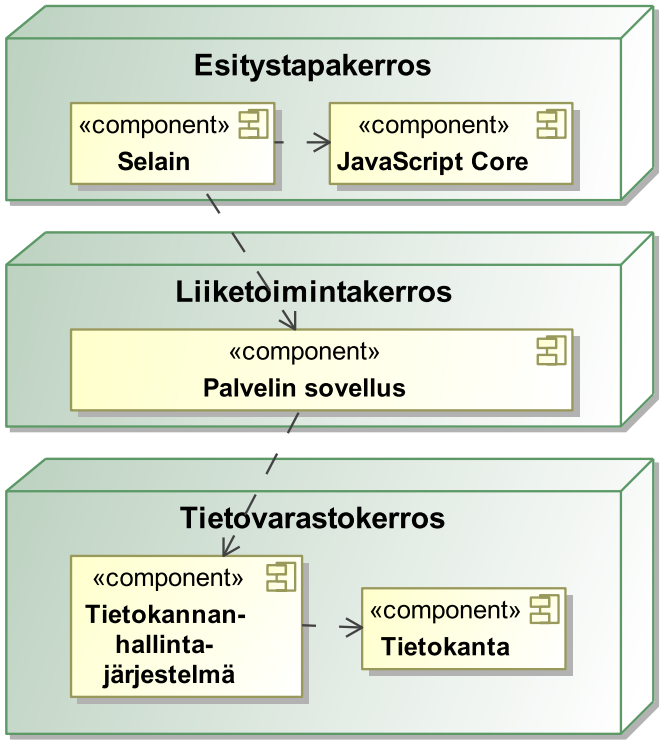
- Yksi asiakkaan vaatimuksista oli yleisesti hyväksyttyjen standardien käyttö, jotta palvelu on mahdollisimman laajasti selaimien tukema ja toisaalta mahdollisimman pitkään tekniikan ja standardien kehittyessä taaksepäin yhteensopiva. Tähän liittyy myös vaatimus nykyaikaisten tekniikoiden käytöstä.

4.1.2 Arkkitehtuuri ja moduulit

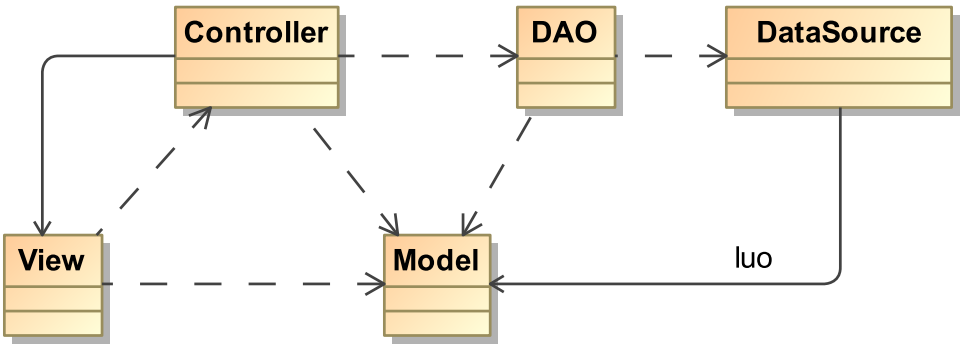
Järjestelmän korkeantason arkkitehtuuri on asiakas-palvelin-arkkitehtuuri. Käyttäjien selaimen esittämä käyttöliittymä on asiakas ja palvelimella ajettava sovellus toimii käyttöliittymälle palvelimena. Laitearkkitehtuuriltaan järjestelmä on myös kolmikerrosarkkitehtuuri, vaikkakin tietokanta ja palvelinsovellus ovat samalla palvelimella. Selaimen käyttöliittymä muodostaa esitystapakerroksen, palvelimen websovellus liiketoimintakerroksen, ja järjestelmäkokonaisuuden integrointipisteenä toimiva tietokannanhallintajärjestelmä toimii tietovarastokerroksena.

Esitystapakerroksen ja liiketoimintakerroksen välinen rajapinta on perinteisesti web-sovelluksessa HTTP-protokollan päällä toimiva HTML- ja XML-kuvaukseen perustuva esitys. Liiketoimintakerroksen ja tietovarastokerroksen välinen tietoliikenne käyttää TCP/IP-protokollapinoa. Palvelinsovellus keskustelee tietovaraston kanssa käyttäen JDBC-rajapintaa. Kuvassa 4.1 on esitetty järjestelmän kolmikerrosarkkitehtuuri.

Sovelluksen ohjelmistoarkkitehtuuri perustuu MVC-arkkitehtuurimalliin ja DAO-suunnittelumalliin. Kuvassa 4.2 on esitetty sovelluksen ohjelmistoarkkitehtuuri. MVC-mallin käsittelijät käyttävät DAO-luokkia tietomallien hakemiseen. DAO-luokat käyttävät sovelluskehyksen niille tarjoamia tietovarastopalveluita tietojen hakemiseen. *DataSource*-rajapinnan taakse kapseloitu tietovarasto luo malliolioita ja palauttaa ne DAO-luokille, jotka välittävät mallit eteenpäin käsittelijöille. Käsittelijät kytkevät mallit näkymiin ja tarjoavat näkymille epäsuorasti palveluita. Tietomallin mukaisia malleja kutsutaan sovelluksessa entiteeteiksi. Tarkempi kuvaus entiteeteistä löytyy alakohdasta 4.2.1. Käsittelijöiden ja näkymien suunnittelusta kerrotaan tarkemmin alakohdassa 4.2.2



Kuva 4.1: Kolmikerrosarkkitehtuuri



Kuva 4.2: Ohjelmistoarkkitehtuuri

4.1.3 Virhe- ja poikkeusmenettelyt

Sovelluksen yleinen poikkeusmenettely on, että poikkeuksia ei heitetä. Arkkitehtuurillisesti matalassa sovelluksessa poikkeukset tuovat turhaa monimutkaisuutta ja lisäävät virheiden mahdollisuuksia toimintalogiikkakerroksella. Kuten Joel Spolsky on artikkelissaan *Joel on Exceptions* todennut, poikkeukset lisäävät ohjelmakoodin kompleksisuutta moninkertaistamalla mahdollisten suorituspolkujen lukumäärän, kun yksittäisten koodilohkojen poistumiskohdat lisääntyvät [35]. Koodia tarkastelemalla on lisäksi vaikea ellei mahdoton nähdä mihin kontrolli siirtyy kunkin poikkeuksen kohdalla.

Järjestelmä- ja kolmannen osapuolen kirjastoista vuotavat poikkeukset käsitellään paikallisesti, ja virheisiin pyritään reagoimaan niin, ettei käyttäjän työnkulku häiriinny. Kaikki poikkeukset otetaan kiinni viimeistään ennen sovelluspalvelimen web-säiliötä, jotta poikkeukset saadaan tallennettua lokiin kontrolloidusti ja käyttäjä ohjattua virhe-sivulle. Poikkeuksista tulostetaan lokiin vähintään kutsuketju ja virheen tapahtumahetki.

Käyttöliittymäsuunnittelussa virheiden käsittely otetaan huomioon suunnittelemalla käyttöliittymät niin, että virheellisten valintojen teko pyritään estämään käyttämällä käyttöliittymäelementtien piilottamista ja käytön estoa. Toisaalta virheellisistä syötteistä pyritään ilmoittamaan reaaliaikaisesti, jolloin käyttäjä voi esimerkiksi huomata virheellisen syötteen jo ennen lomakkeen hyväksymistä.

4.2 Suunnittelu

Tässä kohdassa kuvataan, kuinka järjestelmälle asetetut vaatimukset täyttävät toiminnot suunniteltiin ja millaisia käyttöliittymäratkaisuja näkymät sisältävät. Ensiksi kuvataan tapa, jolla liiketoimintakerrokselle haetaan tietovarastosta tietoa ja miten sitä käytetään. Seuraavaksi tutustutaan tapaan, jolla sovellus saadaan toimimaan web-ympäristössä, eli miten toiminnot tarjotaan käyttäjille ja kuinka käyttäjien istunnot hallitaan. Lopuksi käydään läpi muutama tekniikoiden ja suunnittelun kannalta mielenkiintoinen toimintokokonaisuus käyttöliittymäesimerkkeineen.

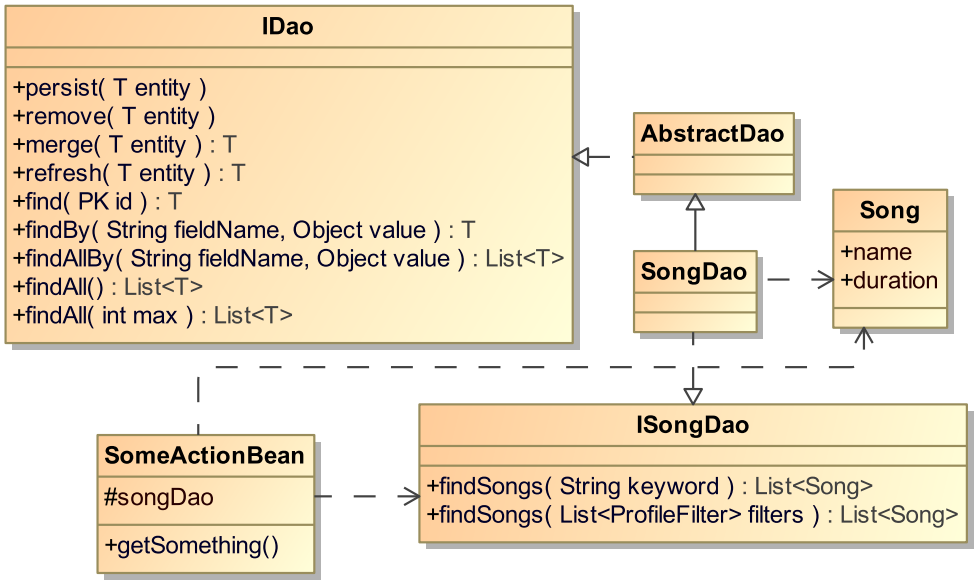
4.2.1 Entiteetit ja niiden käyttö

Entiteetit ovat malli-luokkien olioita. Olio-relaatio muunnoksen ydin on sovelluksen käyttämän olio-pohjaisen tietomallin, eli entiteettien ja niiden välisten suhteiden kuvaus. JPA-standardi mahdollistaa entiteettien kuvaamisen käyttäen yksinkertaisia Java-luokkia, joihin annotaatioita käyttäen kuvataan luokan ominaisuudet. Esimerkiksi järjestelmän käyttämät musiikkikappaleet on mallinnettu Java-luokkana *Song*, jolle on kuvattu annotaatioita käyttäen luokkaa vastaava käsite relaatiomallissa. Annotaatioilla voidaan kuvata entiteetin suhteet muihin entiteetteihin, ominaisuuksien validointiin liittyviä seikkoja ja tietotyyppien tarkennuksia. *Song*-luokalle voisi esimerkiksi annotaatioita käyttäen kuvata, että sillä on suhde albumiin johon tämä kuuluu ja artistiin tai artisteihin, jotka ovat kappaleen esittäjiä. Luokalle voisi myös kuvata, että nimi ja kesto ovat pakollisia ominaisuuksia.

Sovellus käyttää entiteettejä JPA-toteutuksen, eli EclipseLinkin ja Spring-sovelluskehyksen palveluiden avulla. Spring huolehtii tietokantatransaktioista ja sovellukselle tarjotuista palveluista ja käyttää JPA-toteutusta näiden palveluiden suorittamiseen. DAO-suunnittelumallin mukaiset DAO-luokat käyttävät Spring-kehyksen tarjoamia palveluita esimerkiksi näin:

```
String queryString = "select x from Song x where x.artist = :artist";
Query query = entityManager.createQuery(queryString);
query.setParameter("artist", "Eric Clapton");
List<Song> songs = query.getResultList();
```

Esimerkissä *entityManager*-olio on Spring-kehyksen tarjoama palvelu, jonka avulla kyselyt tietokantaan tehdään. DAO-luokat periytetään yhdestä geneerisestä pohjaluokasta, joka tarjoaa peruspalvelut muille luokille. Periyttämällä tämä geneerinen luokka saadaan yhden malliluokan tietovarasto-operaatiot tarjoava luokka. Kuvassa 4.3 on luokkakaavio, jossa on kuvattu *Song*-entiteetit käsittelevän *SongDAO*-luokan hierarkia. Kuvan *SomeActionBean* on esimerkki Stripes-käsittelijästä (katso kuva 4.2).



Kuva 4.3: Esimerkki DAO-mallista

4.2.2 Toiminnot ja näkymät

Varsinainen toimintalogiikka sisältyy MVC-mallin käsittelijöihin, joita ovat Stripes-kehyksen ActionBeanit. Kutakin määrittelyssä kuvattua sivua tulee vastaamaan yksi yhteen URL:ään kytketty ActionBean. ActionBeaneille voidaan luoda yksi yhteinen kantaluokka, jonka avulla näkymät saadaan kytkettyä sovelluksen tilanhallintaan käyttäen ActionBeanContext-luokkaa. Stripes kehyksen *SpringBean*-annotaatioilla käsittelijöihin saadaan injektoitua tietovarastokerroksen palvelut DAO-luokkien avulla.

Kutakin käsittelijää vastaa yksi tai useampi näkymä. Näkymät ovat JSP-sivuja, jotka mahdollistavat entiteettien ja muiden Java-olioiden käsittelyn näkymien merkkauskielen seassa. Stripes tukee näkymien koostamista hierarkkisista näkymäkomponenteista, jolloin kaikille sivuille samat komponentit saadaan helposti lisättyä joka sivulle ja varsinaiset sisältöosiot erotettua muista komponenteista. Näkymien koostaminen toisista näkymistä helpottaa myös vaatimuksena ollutta sisällön uudelleenorganisointia.

4.2.3 Tilanhallinta

Tilanhallinnalla tarkoitetaan tässä yhteydessä sovelluksen autentikointia, autentikoidun käyttäjän tilan, eli istunnon ylläpitoa ja käyttäjän oikeuksien tarkastuksia (autorisointi). Käyttäjän istunnon hallinta perustuu Stripesin ActionBeanContext-luokasta periytettyyn luokkaan. Stripes huolehtii, että kullakin istunnolla on oma olio tuosta luokasta ja näin tuon luokan taakse saadaan kapseloitua istunnon hallinnalle olennaiset asiat.

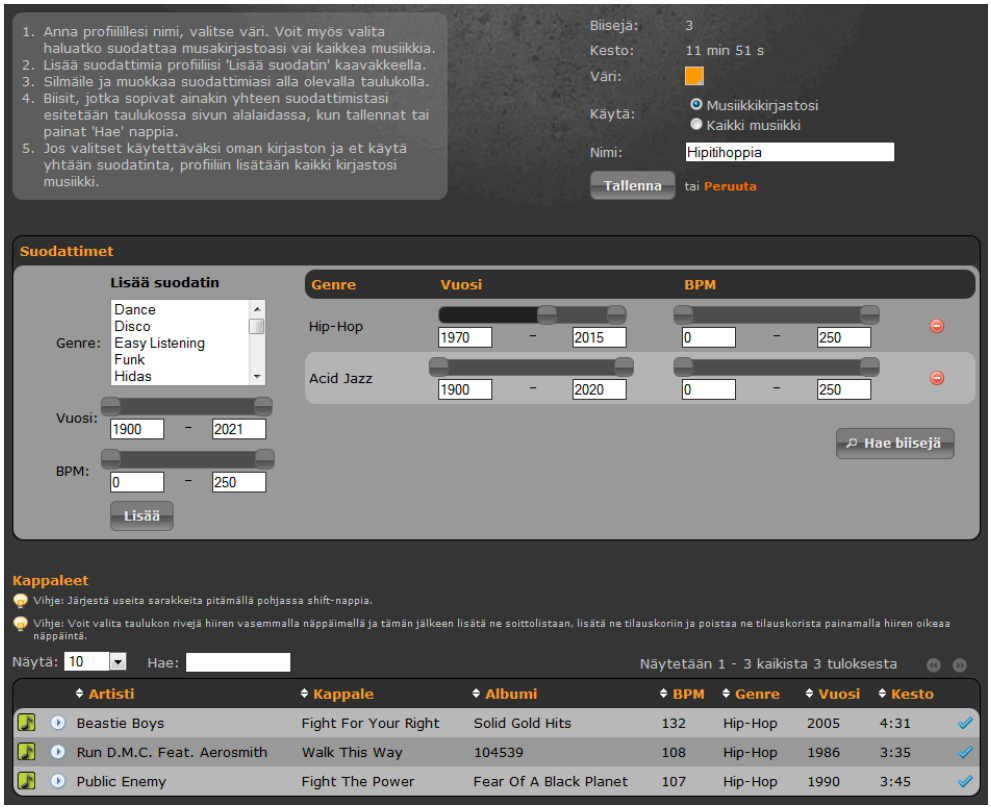
Kaikki palveluun tulevat HTTP-pyynnöt kulkevat BeforeInterceptor-luokan kautta, joka tarkastaa onko istunnolle tallennettu käyttäjään liittyvät asiakastiedot vai ei, josta voidaan päätellä onko käyttäjä kirjautunut sisään. Jos käyttäjä ei ole kirjautunut, hänet ohjataan kirjautumissivulle. Jos kirjautuminen onnistuu, käyttäjään liittyvät asiakas- ja käyttäjätunnisteet tallennetaan istunnon parametreihin. Tämän lisäksi istunnon tietoihin tallennetaan kirjautumishetkellä ulkoisesta tiedostosta poimitut asetukset ja istunnon elinkaareen liittyvä kuuntelija (SessionBindingListener). Kuuntelijan tehtävä on ylläpitää tietokannassa tieto siitä ketkä käyttäjät ovat kirjautuneena sisään.

Käyttäjien oikeudet palvelussa määräytyvät tietokantataulusta haetun järjestelmätyypin mukaan. Palvelupaketti (alakohta 3.1.2) ilmaisee mikä on käyttäjän taustamusiikkisoittimen tyyppi ja järjestelmän oikeudet on määritelty noiden soitintyyppien perusteella. Soittimien tyypit on määritelty järjestelmässä autorisointirooleina, ja roolien tarkastuksista huolehtii SecurityManager-luokka, jolta kaikki ActionBeanit ja JSP-sivut, joilla käyttöoikeuksia on rajoitettu, ne tarkastavat.

4.2.4 Soittoprofiilit

Soittoprofiilien luonti -näkymä laadittiin asiakkaan toimittaman käyttöliittymähahmotelman pohjalta. Profiilit koostuvat suodattimista, ja ideana on, että profiilin muodostamalle soittolistalle päätyy kappaleet, jotka läpäisevät ainakin yhden suodattimista. Yksi suodatin koostuu musiikkigenrestä, alku- ja loppuvuodesta sekä alku- ja lopputahdista. Profiilin lisätyt suodattimet esitetään taulukossa, jossa niitä voi muokata tai poistaa.

Näkymän ylälaidassa on käyttöä helpottamaan laadittu lyhyt opastus profiilin luontia varten ja kaavake profiilin tallennusta varten. Profiilille pystyy valitsemaan sitä kuvaavan värin, antamaan nimen ja valitsemaan halutaanko profiili luoda käyttäjän oman mediakirjaston sisällöstä vai palveluntarjoajan kaikesta musiikkisisällöstä. Genret ladataan Ajaxia käyttäen listaksi suodattimien luontikaavakkeeseen mediasisällön valinnan mukaan. Ajax-ohjelmointityyliä käyttäen ladataan myös soittoprofiiliin täsmäävät kappaleet sivun alalaidassa olevaan taulukkoon. Taulukossa on lisäksi toiminnot, joilla mediakirjastosta puuttuvat kappaleet voidaan lisätä tilauskoriin. Kuvassa 4.4 on esitetty profiilieditorin käyttöliittymä.



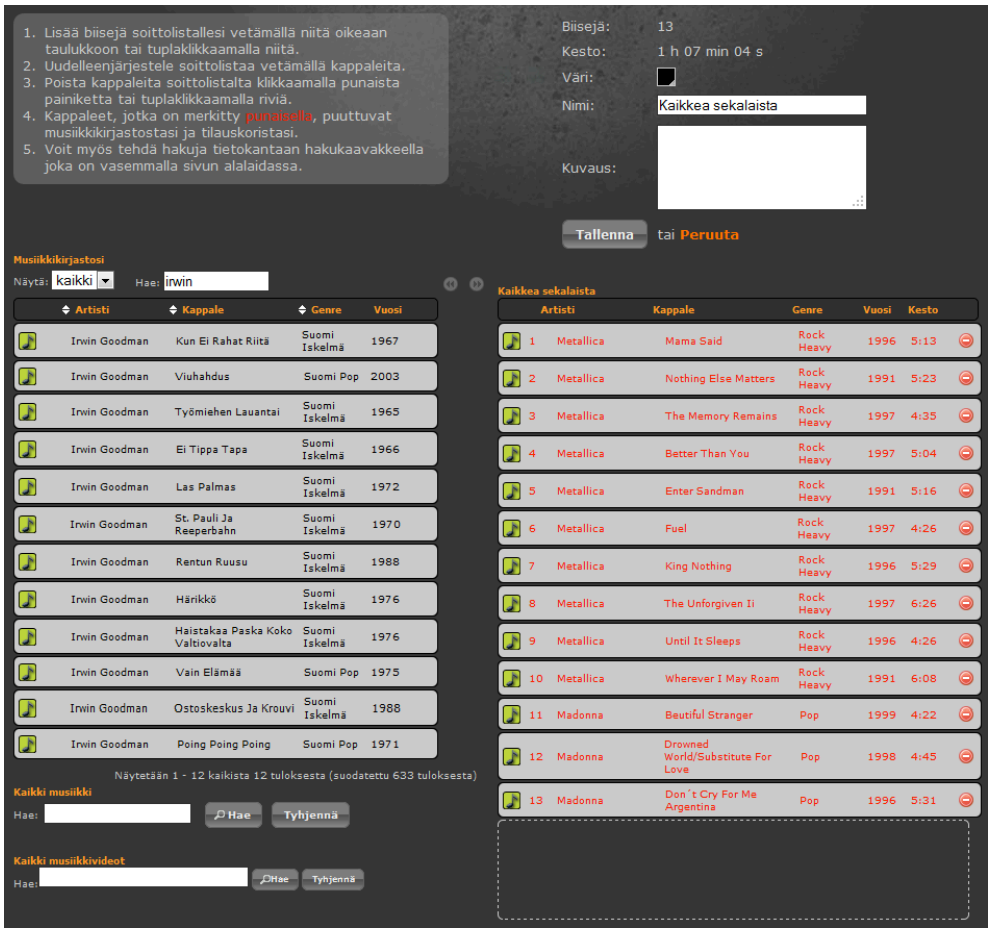
Kuva 4.4: Profiilieditori

4.2.5 Soittolistaeditori

Soittolistaeditori laadittiin niin ikään asiakkaan käyttöliittymähahmotelman pohjalta. Näkymässä on ylhäällä profiilien luonti -sivun tapaan lyhyt opastus soittolistaeditorin käyttöä varten ja tallennuskaavake, jolla soittolista voidaan tallentaa. Soittolistalle voidaan valita väri ja nimi.

Tallennuskaavakkeen alapuolella on lista soittolistalle valituista kappaleista sivun oikeassa laidassa. Soittolistan vasemmalla puolella on käyttäjän musiikkikirjaston sisältö, josta kappaleita voi vetämällä lisätä soittolistalle. Paremman käytettävyyden saavuttamiseksi kappaleen voi lisätä soittolistalle myös kaksoisosoituksella.

Musiikkikirjasto-taulukon alapuolella on hakukaavakkeet, joilla voidaan Ajax-ohjelmointitekniikkaa käyttäen tehdä hakuja palveluntarjoajan mediavalikoimaan. Hakukaavakkeilla voidaan hakea musiikki ja musiikkivideoita. Kappaleita voidaan järjestää soittolistalla vetämällä. Soittolistalta poistaminen tapahtuu kaksoisosoituksella tai painamalla miinus-merkkiä muistuttavaa kuvaketta. Kun soittolistan tallentaa, järjestelmä kysyy käyttäjältä haluaako tämä lisätä mediakirjastosta puuttuvat kappaleet tilauskoriin. Mediakirjastosta puuttuvat kappaleet näkyvät soittolistalla punaisella tekstillä. Kuvassa 4.5 on esitetty soittolistaeditorin käyttöliittymä.

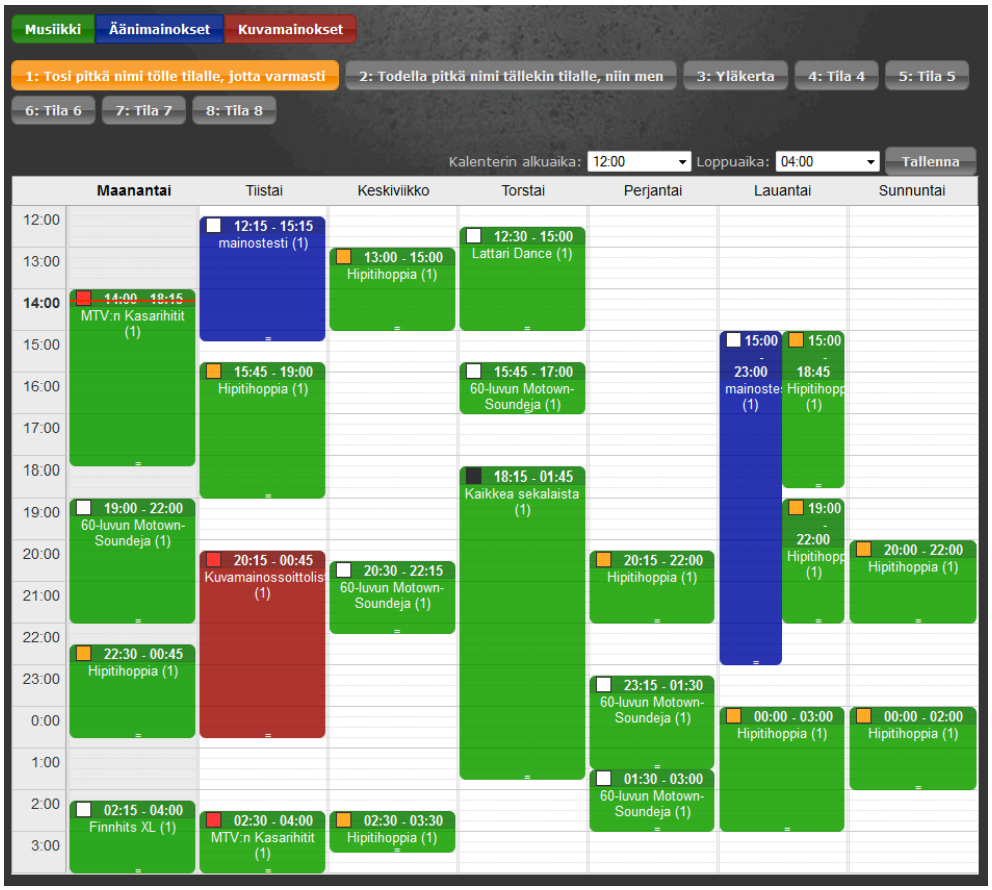


Kuva 4.5: Soittolistaeditori

4.2.6 Ajastukset

Ajastukset-sivun pohjana oli ajatus mahdollisesta viikkokalenterinäkymästä. Tiedossa oli, että JavaScriptillä on mahdollista luoda hyvin selaimessa toimiva viikkokalenterinäkymä, mutta varmaa ei ollut, olisiko ajastukset helppo ja nopea luoda tällaista näkymää käyttäen. Pienellä kokeilulla viikkokalenteri osoittautui toteutuskelpoiseksi ideaksi, ja sen pohjaksi valittiin kolmannen osapuolen tekemä kalenterikomponentti, jota laajennettiin tukemaan halutut toiminnot.

Ajastuksia voidaan luoda vetämällä tai osoittamalla kalenteriin uusi ajastus, jonka jälkeen aukeaa dialogi, johon ajastuksen tarkemmat tiedot syötetään. Olemassa olevia ajastuksia voi siirtää tai venyttää vetämällä ja muokata osoittamalla, jolloin käyttäjälle aukeaa sama dialogi kuin lisättäessä ajastusta. Näkymän ylälaitaan lisättiin lisäksi painikkeet, joilla voidaan rajata esitettävät ajastukset tilojen ja mediatyypin perusteella. Kuvassa 4.6 on esitetty ajastuksien käyttöliittymä. Kuvassa on valittu tila numerolla yksi ja kyseisen tilan kaikkien mediatyyppien ajastukset.



Kuva 4.6: Ajastukset

4.3 Toteutus ja testaus

Tässä kohdassa kuvataan millaisia projektinhallintamenetelmiä käytettiin edellä kuvattujen toimintojen toteutuksessa ja kuinka jatkuvat integrointi ja testaus olivat osa ohjelmistokehitystä.

4.3.1 Ketterät toteutusmenetelmät

Projektin alusta asti oli selvää, että asiakas haluaa olla tiiviisti läsnä toteutuksen suunnittelussa ja vaikuttaa projektin etenemiseen. Projektinhallintaan päätettiin

käyttää Scrum-menetelmää, joka on iteratiivinen ja inkrementaalinen menetelmä projektin toteutuksen tehostamiseen [33]. Scrumin ideana on toimia projektin runkona, joka määrittelee projektiin osallistuville henkilöille roolit ja ohjeet kuinka missäkin roolissa toimitaan. Rooleja ovat tuoteomistaja, ScrumMaster ja Scrum-tiimin jäsen. Menetelmä koostuu yksinkertaisimmillaan tuotelistasta, työlistasta, vaatimuksista ja sprinteistä, eli iteraatioista.

Tuoteomistaja laatii ja järjestää tuotelistan, joka koostuu kaikista järjestelmän vaatimuksista. Tuotelista on tarkoitus pitää kokoajan järjestettynä niin, että listan alusta löytyvät asiakkaalle kullakin hetkellä tärkeimmät vaatimukset. Varsinaisen ohjelmistokehityksen tekevät Scrum-tiimit, jotka koostuvat yleensä noin viidestä henkilöstä. Tiimejä ja tuoteomistajaa auttaa ScrumMaster, joka vastaa tiimien työrauhasta ja auttaa tiimejä tunnistamaan ja selvittämään ongelmia. Kullekin iteraatiolle eli sprintille laaditaan jokaista tiimiä kohden oma työlista, johon tiimin jäsenet valitsevat sen määrän vaatimuksia, jonka he uskovat sprintin aikana pystyvänsä toteuttamaan. Vaatimukset valitaan työlistalle sprintin aluksi pidettävässä palaverissa. Kun vaatimukset valitaan tuotelistan alusta, ensimmäiseksi työn alle päätyvät asiakkaalle tärkeimmät vaatimukset. [33]

Täysin orjallisesti Scrumia ei noudatettu, eikä se ollut tarkoituskaan. Iteraatioiden pituudeksi valittiin kaksi viikkoa, ja asiakas saapui kahden viikon välein sprintin aluksi järjestettävään palaveriin. Palaverissa esiteltiin edellisen sprintin tuotokset, saatiin palautetta järjestelmästä ja valittiin sprintin työlistalle iteraatiossa toteutettavat vaatimukset. Virallista roolijakoa ei projektissa käytetty, vaikkakin roolijako muodostui omalla painollaan hyvin Scrumin roolijakoa vastaavaksi.

4.3.2 Yksikkötestaus

Yksikkötestaus, jota joskus myös kehittäjätestaukseksi kutsutaan, päätettiin tehdä niin, että tärkeimmät luokat ja erityisesti MVC-mallin käsittelijät testataan yksikkötestein. Sovelluskehittäjille ajan myötä usein kehittyy kyky tunnistaa mille ohjelmakoodille kannattaa yksikkötestit kirjoittaa ja mille se on hyvin suurella todennäköisyydellä turhaa. Usein projekteissa sovitaan jokin testikattavuus johon pyritään, mutta tällä kertaa päätettiin jättää testaus kehittäjien oman harkinnan varaan.

Automatisoituja testejä varten laadittiin asetukset, jotka mahdollistivat testien ajamisen ajonaikaisesti luotua muistitietokantaa vasten. Tietokanta luotiin entiteetti-kuvausten perusteella JPA-toteutuksen avustuksella. Käyttämällä oikeaa tietokantaa, käsittelijöiden testaus helpottuu suuresti, kun kaikkia käsittelijöiden käyttämiä palveluita ei tarvitse simuloida tehden työläitä tynkä-toteutuksia. Usein oikean tietokannan käyttö myös tuo esiin virheitä, jotka tynkiä vasten testattuna jäisivät huomaamatta.

Testit toteutettiin TestNG-testeinä, ja niille laadittiin hierarkia, jossa ensimmäi-

seksi ajettiin yksikkötestit, joista saatiin ulos käyttäjäistunto. Istuntoa voitiin käyttää hyödyksi muissa käsittelijöitä testaavissa testeissä, sillä käsittelijät oli suunniteltu niin, että ne aina tarvitsevat käyttäjäistunnon. Tärkeä kriteeri automatisoiduissa testeissä oli, että ne saatiin helposti ajettua kehitysympäristössä ja Jenkins CI -työkalulla integrointiympäristössä jatkuvan integroinnin periaatteiden mukaisesti.

4.3.3 Järjestelmätestaus

Usein järjestelmätestaus on hankala automatisoida, mutta nykypäivänä erityisesti web-sovelluksilla on saatavilla erilaisia työkaluja, joilla web-palvelua voidaan järjestelmätestata. Sovelluksen järjestelmätestit tehtiin käyttäen Selenium-sovelluskehystä ja testit pyrittiin laatimaan sopivin etapein ohjelmistokehityksen ohessa. Liian aikaisin laaditut järjestelmätestit kuitenkin osoittautuivat työläiksi ylläpitää. Toisin kuin yksikkötestien vastinkappaleet, eli luokkien rajapinnat, järjestelmätestien vastinkappaleet, eli käyttöliittymät, ovat hyvin alttiita muutoksille kehitysvaiheessa.

Automatisoidut järjestelmätestit ovat hyviä tunnistamaan komponenttien integroinnista aiheutuvia ongelmia ja ne toimivat loistavasti perusominaisuuksien regressiotestauksessa. Niillä ei kuitenkaan voi korvata manuaalista järjestelmätestausta, sillä ne eivät voi tunnistaa kosmeettisia virheitä, eikä niillä voi saada kovinkaan suurta kattavuutta koko sovelluksen toiminnoista. Perinteisesti järjestelmätestit laaditaan asiakasvaatimuksia vasten, automatisoidut testit laaditaan kuitenkin laatimishetkellä olevaa sovelluksen toimintaa vasten. Tästä seuraa, että testit läpäisevä järjestelmä ei välttämättä vastaa asiakkaan vaatimuksia.

Projektiin hyvin aktiivisesti osallistunut asiakas hoiti osan manuaalisesta järjestelmätestauksesta itse. Tällainen järjestely sopi loistavasti projektiin, koska näin asiakkaan ei tarvinnut käyttää resursseja kaikkien vaatimusten yksityiskohtaiseen kuvaamiseen testausta varten.

4.3.4 Käytettävyyden testaus

Käytettävyyden testauksella tarkoitetaan tässä yhteydessä ohjelmiston helppokäyttöisyyden, opittavuuden, muistettavuuden ja käyttäjälle sallittujen virheiden määrän testausta. Sallittujen virheiden määrällä tarkoitetaan virheitä, joita käyttöliittymä sallii käyttäjän tehdä. Esimerkiksi monesti postinumero syötetään merkkijonona, jolloin postinumero-kenttään on mahdollista kirjoittaa vaikkapa "abcde"ja vasta kun lomake validoidaan, käyttäjälle ilmoitetaan virheellisestä syötteestä. Sallittujen virheiden määrää voitaisiin vähentää esimerkin tapauksessa estämällä muiden kuin numeroiden syöttäminen kyseiseen kenttään, vaikkakin kenttä luettaisiin järjestelmään merkkijonona. Toisin kuin ohjelmistoa yleisesti, käytettävyyttä on hankala

testata toistettavin testein eivätkä mahdolliset ongelmat ole luonteeltaan niin ilmeisiä kuin ohjelmavirheet sovelluksessa. Esimerkiksi puna-vihersokean voi olla vaikea erottaa värein koodatut napit toisistaan. Tämän testaaminen ei ole helppoa, sillä värisokeus on hyvin yksilöllistä. Käytettävyyttä voidaan kuitenkin testata, ja käyttökokemuksen saadessa yhä enemmän ja enemmän painoarvoa tämän päivän ohjelmistokehityksessä, on se oleellinen osa testausta.

Käytettävyyden testauksella pyritään löytämään ongelmia sovelluksen käyttöliittymästä ja toimintalogiikasta. Sovelluksen tulisi olla johdonmukainen läpi näyttöjen ja toimintojen ja toisaalta johdonmukainen muihin vastaaviin sovelluksiin verrattuna. Käyttöliittymässä käytetyillä väreillä tulisi olla merkitys ja käyttäjän toimia tulisi ohjata toisteisin vihjein, eli esimerkiksi napissa voisi tekstin lisäksi käyttää kuvaketta.

Testausta voi tehdä lukuisin eri menetelmin, mutta projektissa valittiin käytettäväksi joitakin helpoiten järjestettävistä tekniikoista. Kustannustehokkain menetelmä on käyttöliittymäsuunnittelijoiden ja kehittäjien muun kehitystyön ohella tekemä jatkuva käytettävyyden tarkastelu. Tarkastelua varten on olemassa heuristiikkoja, jotka ovat eräänlaisia käytettävyyden auktoriteettihenkilöiden laatimia tarkastuslistoja. Heuristiikkaa läpi käymällä käyttöliittymästä on helppo havaita ongelmia, jotka muuten jäisivät ehkä huomaamatta. Esimerkiksi epätasaisen levyiset tekstikentät saavat kaavakkeen näyttämään sekavalta, ja voivat antaa käyttäjälle väärän kuvan siitä, mitä kenttään pitää syöttää. Projektin toteutuksen tukena käytettiin muun muassa tunnettuja Nielsenin heuristiikkoja [30].

Käytävätestaus on myös helposti järjestettävä ja tehokas menetelmä. Käytävätestauksessa pyydetään projektin ulkopuolista henkilöä käyttämään sovellusta, tarkkaillaan tämän reaktioita ja kuunnellaan tämän kommentteja ja huomioita sovelluksesta. Menetelmä on tehokas, mutta sitä ei voi soveltaa kaikkiin sovelluksiin. Jos järjestelmä on ammattilaiskäyttöön tarkoitettu ja kohdekäyttäjillä on erityinen tuntemus sovellusalueeseen, ei kenen tahansa huomiot välttämättä ole arvokkaita, vaan ne voivat olla jopa harhaanjohtavia. Toisaalta testaavan henkilön henkilökohtaiset mieltymykset saattavat vaikuttaa kokemukseen. Käytävätestausta suoritettiin läpi projektin muutaman iteraation välein, kun uusia näkymiä oli valmistunut testattavaksi. [29]

Arvokkainta mutta hankalimmin järjestettävää testausta on järjestelmän todellisen käyttäjän kanssa tehty käyttäjätarkkailu. Tällainen käyttäjätarkkailu voidaan järjestää esimerkiksi pilottiasiakkaan kanssa sopivassa välietapissa tai projektin loppupuolella. Käyttäjätarkkailun ja käytävätestauksen istunnot kannattaa tallentaa videonauhalle, jotta käyttäjän reaktiot ja kommentit saadaan talteen vääristymättä esimerkiksi muistiinpanoissa. Käyttäjätarkkailu järjestettiin niin, että käyttäjänä oli yksi musiikkialan ammattilainen ja tarkkailua tekivät asiakkaan edustaja ja osa ke-

hittäjistä. Testikäyttäjää pyydettiin käyttämään ääneen ajattelu -menetelmää, jossa käyttäjä kertoo ääneen järjestelmän käytössä kohtaamistaan ongelmista, yllätyksistä ja erityisen hyvistä ratkaisuista. Käyttäjälle ei tulisi kuitenkaan jutella, eikä häntä saisi auttaa ongelmien kanssa. [36]

4.4 Käyttöönotto

Käyttöönottoon liittyviä toimenpiteitä projektissa olivat lopullisen ohjelmakoodin ja dokumenttien toimitus asiakkaalle sekä asiakkaan tuotantoympäristön pystytys. Ohjelmakoodi toimitettiin asiakkaalle myös jokaisen iteraation päätyttyä. Asiakkaan tuotantoympäristön pystytys aloitettiin hyvissä ajoin ennen projektin päättymistä, jotta mahdolliset ongelmat saatiin esiin mahdollisimman varhaisessa vaiheessa. Vaikka asiakkaan ympäristö oli hyvin vastaava kuin projektin toteutuksessa käytetty jatkuvaa integrointia varten pystytetty ajoympäristö, ongelmia esiintyi silti. Isoimmat ongelmat johtuivat eroista tietokannoissa, mutta myös sovelluspalvelinten väliset erot teettivät työtä.

Ennen varsinaista käyttöönottoa järjestelmä pyritään ottamaan pilottikäyttöön yhden loppukäyttäjän kanssa. Pilotoinnilla voidaan varmistaa monia asioita, joiden testaaminen muuten on hyvin hankalaa. Tärkeitä varmistettavia asioita on muun muassa resurssien riittävyys, eli onko tuotantopalvelimessa tarpeeksi muistia, ovatko tietoliikenneyhteydet riittävän nopeita, toimivatko välimuistit ja muut optimoinnit kuten on tarkoitettu ja niin edelleen. Tietokannan eheys on myös yksi varmistettava asia. Eheydellä tarkoitetaan tässä yhteydessä sitä, ettei tieto korruptoidu toisten järjestelmien näkökulmasta. Karkea esimerkki voisi olla, että toinen järjestelmä asettaa joihinkin monikoihin tietylle ominaisuudelle tyhjäksi arvoksi *null*-arvon ja toinen järjestelmä taas olettaa tyhjän arvon olevan tyhjä merkkijono.

Pilotoinnista saadut ensimmäiset oikeat käyttökokemukset ovat myös tärkeitä ja mahdollistavat mittavienkin muutosten tekemisen vielä ennen varsinaista käyttöönottoa. Tarkoitus on pitää vanha järjestelmä rinnakkaisessa käytössä pilotointivaiheen ajan, jolloin ongelmatilanteessa loppukäyttäjät voivat turvautua vanhaan järjestelmään. Käyttöönottovaiheen päätyttyä projektikin päättyy ja ohjelmistokehitykseen osallistuneet henkilöt voidaan sijoittaa uusiin projekteihin ja siirtyä järjestelmän ylläpitovaiheeseen.

4.5 Ylläpito

Ylläpito tarkoittaa ohjelmiston elinkaaren vaihetta, jossa järjestelmää käytetään suunnitellulla tavalla. Projektin asiakas itse huolehtii ohjelmiston jatkokehityksestä ja ohjelmavirheiden korjaamisesta. Asiakkaalle tarjotaan kuitenkin tarvittaessa tukea teknisten ongelmien kanssa. Ylläpitovaihe kestää koko ohjelmiston käytön ajan,

eli mahdollisesti useita vuosia.

Ylläpito-ohje on tärkeä osa ylläpitovaihetta ja hyvään ylläpito-ohjeeseen panostamalla vältetään tilanteet, joissa yksinkertaisen ongelman ratkaisemiseksi asiakas joutuu ottamaan järjestelmän toimittajaan yhteyttä. Tärkeää onkin aloittaa ylläpito-ohjeen laatiminen hyvissä ajoin projektin kuluessa, jotta sinne voidaan kerätä huomioita ja kokemuksia esimerkiksi integrointi-, testaus- ja tuotantoympäristöjen pystytyksestä. Toisaalta on tärkeää kirjata myös käyttöönottoon ja ylläpitoon liittyvät kokemukset talteen tulevien projektien varalta.

5. ARVIOINTI

Tässä luvussa on tarkoitus arvioida diplomityön, työn tuloksena saadun järjestelmän ja asiakasyhteistyön onnistumista. Myöhemmin tästä kokonaisuudesta käytetään nimitystä projekti. Luvussa käydään läpi järjestelmän tämänhetkinen tila ja tarkastellaan järjestelmää sille asetettujen tavoitteiden näkökulmasta. Täyttääkö järjestelmä sille asetetut tavoitteet, jäikö jotain tekemättä tai olisiko jotain pitänyt tehdä toisin? Seuraavaksi arvioidaan toteutuksen teknistä laatua. Olivatko tekniikka- ja teknologiavalinnat oikeita vai olisiko joku toinen tekniikka pitänyt valita johonkin tarpeeseen? Seuraavaksi käydään läpi asiakkaalta saatua palautetta työprosessista ja toimitetusta järjestelmästä. Lopuksi arvioidaan lyhyesti millainen tulevaisuus järjestelmällä on mahdollisesti edessä ja annetaan kehitysehdotuksia.

5.1 Tilanne kirjoitushetkellä

Kun työ aloitettiin, sen pääasiallinen tavoite oli yhdessä asiakkaan kanssa määritellä, suunnitella ja toteuttaa järjestelmä, joka täyttää asiakkaan sille asettamat tavoitteet. Toisaalta tarkoitus oli myös toteuttaa projektin aikana syntyneet ideat ja niiden pohjalta laaditut uudet tavoitteet. Tärkeitä tavoitteita olivat lisäksi asiakkaalle luvattu sataprosenttinen tyytyväisyystakuu ja toteutuksen ehdottoman hyvä tekninen laatu. Kirjoitushetkellä projekti on käyttöönottovaiheessa. Järjestelmätestausta tehdään vielä ja järjestelmään on tarkoitus tehdä joitakin kosmeettisia muutoksia. Lähiviikkoina olisi tarkoitus aloittaa pilotointi tarkoitukseen valittujen loppukäyttäjien kanssa.

Käyttöönotto on edistynyt viimeiset kuukaudet melko hitaasti, koska asiakkaan on täytynyt keskittyä muiden järjestelmiensä saattamiseen ajan tasalle. Järjestelmästä on laadittu ylläpitoa varten yhdistetty ylläpito- ja suunnitteludokumentti, joka sisältää kuvauksen teknisestä toteutuksesta. Asiakkaalle on annettu jatkuvaa teknistä tukea, jotta asiakas on saanut pystytettyä kehitysympäristöt järjestelmän jatkokehitystä varten ja saanut järjestelmän ajoon käyttöä varten.

5.2 Tavoitteiden täyttyminen

Projektin voidaan katsoa olevan onnistunut, mikäli sille asetetut tavoitteet ovat täyttyneet. Tavoitteita olivat toiminnallisten- ja yleisten vaatimusten täyttyminen, joustava yhteistyö sekä onnistuneesta projektista saatu kokemus ja oppi. Toiminnallisia

vaatimuksia olivat etukäteen suunnitellut toiminnot, jotka asiakas määritteli alustavin käyttöliittymäkuvin. Vaatimuksiksi lasketaan myös toteutuksen yhteydessä tulleet ideat ja niiden pohjalta laaditut toiminnot, joita ei alunperin oltu suunniteltu.

Kaikki toiminnalliset vaatimukset saatiin toteutettua ja myös sellaiset toiminnot, joiden toteutettavuus ei ollut varmaa, saatiin tehtyä parhaalla mahdollisella tavalla. Yleisiä projektille asetettuja vaatimuksia olivat avoimuus, eli yleisten standardien käyttö, laajennettavuus, luotettavuus, valittujen tekniikoiden modernius, riippumattomuus, skaalautuvuus, suorituskyky ja tietoturvallisuus.

5.2.1 Avoimuus, laajennettavuus ja tekniikkavalinnat

Avoimuuden voidaan katsoa toteutuneen. Vaikka muutama toiminto pohjautuukin hyvin yksilöityihin ratkaisuihin, on kaikki tiedonsiirto ja käytetty notaatio standardien mukaista. Laajennettavuus oli yksi arkkitehtuurin lähtökohdista ja Java EE -ohjelmistoalustaa sekä Stripes-sovelluskehystä käyttäen voidaan todeta, että sovellus on niin hyvin laajennettavissa kuin voidaan sovellusalueen puitteissa tarvita. Luotettavuuteen pyrittiin automatisoidun testauksen, jatkuvan integroinnin, käytettävyyden testauksen ja virhesietoisuuden keinoin. Automatisoiduille testeille ei asetettu varsinaisia kattavuuskriteereitä, mikä laskee hieman niiden vaikutusta luotettavuuteen. Sovelluksen käytössä havaitut virheet tallennetaan kuitenkin lokiin ja näin niiden korjaaminen on mahdollista vaikka käyttäjät eivät ongelmista raportoisikaan. Järjestelmätestauksessa havaitut virheet olivat enimmäkseen kosmeettisia tai johtuivat väärinkäsityksistä asiakkaan ja ohjelmistosuunnittelijoiden välillä. Tarkempaa tietoa luotettavuudesta saadaan vasta todellisessa tuotantokäytössä.

Web-tekniikoissa on erittäin kova kilpailu, mikä osaltaan tekee tekniikoiden valinnan vaikeaksi. Toisaalta hyviä sovelluskehyksiä esimerkiksi Java EE -ohjelmistoalustalle on paljon, eli paljon käytetyn kehyksen valinta tuskin meni pahasti pieleen. Käytetyiksi kirjastoiksi ja sovelluskehyksiksi valittiin aikaisemmissa projekteissa hyviksi todetut ja ympäri maailman yleisesti käytetyt komponentit, kuten Stripes, Spring ja EclipseLink. Komponentit ovat moderneja ja niihin tulee jatkuvasti virhekorjauksia ja uusia ominaisuuksia. Kaikki järjestelmän toteutuksessa käytetyt kolmannen osapuolen ohjelmistokomponentit ovat lisensseiltään sellaisia, että niitä voi ilmaisiksi käyttää kaupallisessa web-sovelluksessa. Järjestelmän voidaan siis todeta olevan moderni ja riippumaton.

5.2.2 Skaalautuvuus ja suorituskyky

Järjestelmän skaalautuvuus ja suorituskyky ovat pitkälti laitteiston varassa. Yksittäinen käyttäjäistunto vie hyvin vähän muistia, eikä käyttäjien lukumäärä muutenkaan merkittävästi lisää järjestelmän resurssienkulutusta. Pullonkaulan skaalautu-

vuudessa voidaan olettaa olevan tietokanta ja tätä varten sovelluksessa käytetään välimuistia, johon olio-relaatio muunnoksesta saadut oliot tallennetaan. Jaetun tietokannan indeksointi on hankalaa, sillä kaikki sovellukset tekevät luultavasti hieman erilaisia kyselyitä tietokannanhallintajärjestelmälle. Jotain indeksejä on kuitenkin mahdollista tehdä ja erityisesti kyselyitä optimoimalla voidaan parantaa suorituskykyä, mikäli se ei ole riittävällä tasolla. Välimuistista johtuen järjestelmä kuluttaa käytännössä kaiken muistin, joka sille annetaan käytettäväksi, mutta muuten laitteistovaatimukset ovat maltilliset ja sovelluksen suorituskyky on hyvä.

5.2.3 Tietoturva

Tietovarastokerroksen tietoturva-asiat ovat tämän projektin ulkopuolella, eli tietoturvatavoitteiden täyttymisen tarkastelu voidaan rajoittaa web-sovellukseen. Web-sovellukselle yleisiä tietoturvauhkia ovat niin sanotut XSS-hyökkäykset (Cross-Site Scripting), DoS-hyökkäykset (Denial of Service) ja tietomurrot. Koska kyseessä on ekstranet-palvelu, on melko epätodennäköistä, että palvelu kerää huomiota Internetissä ilkivaltaa tekeviltä. Tietoturva-asiat tulee kuitenkin ottaa vakavasti. XSS-hyökkäyksien varalta kaikki HTTP-pakettien parametrit suodatetaan mahdollisten merkkaus- tai koodi-injektioiden varalta. Järjestelmässä ei ole julkisia, kaikille avoimia sivuja, eikä julkista rekisteröitymismahdollisuutta ja voidaan olettaa, että maksavat loppukäyttäjät eivät tee järjestelmälle ilkivaltaa. DoS-hyökkäykset voivat siis kohdistua vain sisäänkirjautumissivuun. Mikäli kyseisiä hyökkäyksiä havaitaan, on helppo tehdä kirjautumiseen esto, joka estää kyselyt samasta IP-osoitteesta, jos niitä tulee esimerkiksi yli yksi kappale kahdessa sekunnissa. Lisäksi kaikki liikenne sovelluspalvelimen ja loppukäyttäjien välillä on salattu käyttäen HTTP-protokollan salattua HTTPS-versiota.

5.3 Tekninen laatu

Muiden järjestelmien kanssa jaettu tietokanta oli myös teknisen laadun kannalta suurin haaste. Koska tietomalli oli ennalta laadittu, täytyi sovelluksen mukautua niihin kompromisseihin joita toisten järjestelmien kanssa oli tehty. Joitain muutoksia tietomalliin tehtiin ja lopputuloksena saatiinkin hyvin toimiva olio-relaatio muunnos.

Risto Nevalainen on artikkelissaan *Ketterästi tehtyä ja laadukasta - onnistuuko?* tuonut esiin monia samoja havaintoja, joita tämän projektin onnistumisesta voi tehdä [28]. Ketterästi tekemällä voi onnistua ja kuten monessa muussakin projektissa, tässäkin onnistuminen tuli siitä, että ohjelmisto tehtiin yhdessä asiakkaan kanssa. Nopeatempoisuus ja määrittelyn, suunnittelun ja testauksen päällekkäisyys ei luo mielikuvaa laadusta, mutta todellisuudessa laatu tulee siitä, että keskeisimmät asiakkaan toivomat ominaisuudet toteutetaan järjestelmään mahdollisimman

varhaisessa vaiheessa, ja näin ne kehittyvät ja niitä testataan usean iteraation ajan.

Arkkitehtuurista haluttiin yksinkertainen ja yksinkertaisuuden nimissä siitä jätettiin pois yksi web-sovelluksissa yleisesti käytetty palvelukerros pois. Palvelukerroksen ideana on toimia käsittelijöiden ja tietokantapalveluiden välissä yleisten toimintojen, eli palveluiden, toteutuskerroksena. Projektin alussa tuntui siltä, että palvelukerroksesta kuitenkin tulisi vain läpikutsurajanpinta DAO-luokkiin, ja se päätettiin jättää pois. Jälkeenpäin ajateltuna se olisi kuitenkin ollut hyvä toteuttaa, sillä muutama useassa paikassa toistuva käsittely olisi voitu toteuttaa yhdessä paikassa käyttäen kyseistä palvelukerrosta. Kokonaisuutena arkkitehtuuri on kuitenkin vaatimukset täyttävä.

Järjestelmän testaus oli kehitysvaiheessa pitkälti ohjelmistosuunnittelijoiden itsensä vastuulla. Yksikkötesteille ei asetettu kattavuustavoitteita ja näin testit laadittiin vain niille luokille ja metodeille, joiden katsottiin olevan hyvä testata. Automatisoidut järjestelmätestit laadittiin vain keskeisimmille näkymille ja toiminnoille. Testaukseen olisi voinut käyttää siis paljon enemmänkin resursseja, ja poiketen monista ohjenuorista testauksen työmäärä ei ollut lähelläkään toteutukseen käytettyä työmäärää. Tämä oli kuitenkin tietoinen päätös, ja koska sovellus lähdekoodeineen toimitettiin jokaisen iteraation päätteeksi asiakkaalle, olisivat laatuongelmat tulleet nopeasti esiin, jos niitä olisi syntynyt. Testien parempi kattavuus olisi toisaalta helpottanut uusien ominaisuuksien kehittämistä, mutta toisaalta myös tekstikoodia joutuu ylläpitämään. Testauksen voidaan todeta olleen riittävä, sillä asiakas on ollut tyytyväinen tuotteeseen.

5.4 Saatu palaute

Palautteen saaminen on vaikeaa, kun taloudelliset asiat luovat jännitteitä asiakkaan ja järjestelmän toimittajan välille. Suomalaisen yrityskulttuurin yksi ikävistä piirteistä on positiivisen palautteen puute, sitä ei vain anneta. Realistisen kuvan muodostamiseksi onkin tärkeää pyrkiä saamaan palautetta läpi projektin eikä vain projektin päätteeksi.

Koska projektin asiakas oli itsekin ohjelmistoalan yritys ja yhteistyö oli tiivistä, oli palautteen saaminen odotettua helpompaa. Palautetta tuli iteraatioiden päätteeksi ja projektin päätyttyä saimme asiakkaalta julkaisua varten laaditun lyhyen palautteen [40]:

```
Projekti meni näin suureksi kokonaisuudeksi ihmeellisen hyvin.
Olen tyytyväinen Vincitin tehokkaaseen toteutukseen sekä
erityisesti joustavuuteen projektin määrittelyissä.
Lopputuloksena saatiin juuri sellaista jälkeä kuin haluttiinkin.
```

```
Missään vaiheessa projektia ei kahlittu alkuperäiseen
määrittelyyn vaan palvelu sai elää uusien ideoiden mukana.
Tiimi teki itsenäisiä ratkaisuja ja ehdotuksia tarpeen vaatiessa.

Tekniikan tuntemus ja toteutuksen laatu oli huippuluokkaa.
Myös Vincitin tyytyväisyystakuu oli kiva lisä pakettiin.
Toivon, että saamme jatkossakin yhtä hienoja onnistumisia.
```

Toinen osa palautetta on loppukäyttäjiltä saatu palaute. Jos asiakas on tyytyväinen järjestelmään, mutta käyttäjiltä saatu palaute osoittaa järjestelmässä ongelmia, eivät järjestelmän alkuperäiset tavoitteet ole välttämättä olleet kunnossa. Toisaalta jos käyttäjät raportoivat virheistä paljon tai turhautuvat järjestelmään, on luultavasti kyse laatu- tai käytettävyysongelmista. Koska projektissa tehty järjestelmä ei ole vielä käytössä, ei loppukäyttäjiltä ole saatu palautetta.

5.5 Jatkokehitys

Kirjoittaja uskoo, että kehitys vie tulevaisuudessa tietojärjestelmiä selkeästi kohti kahta pääteemaa. Ensimmäinen teemoista on sovelluksen ajaminen pilvessä. Tällä tarkoitetaan pilvilaskentaa, jossa loppukäyttäjä tai järjestelmän ylläpitäjä ei tiedä ajoympäristön fyysistä sijaintia tai teknisiä yksityiskohtia. Sovellus saattaa todellisuudessa olla hajautettu usealle palvelimelle. Pilvilaskennan etuna on vastuun jakaminen, resurssienkulutukseen pohjautuva laskutus ja resurssien määrä. Kun järjestelmä asennetaan pilveen, ei ylläpidon tarvitse huolehtia laitteistorikoista, palvelimen verkko-ongelmista tai kalliiden laitteiden hankkimisesta. Projektissa toteutetun web-palvelun siirtäminen pilveen olisi melko yksinkertaista, mutta varsinaisesti hyötyä tästä saataisiin vasta, jos asiakkaan muut liittyvät järjestelmät siirrettäisiin myös pilveen.

Toinen teema, jonka mediaa tarjoavat web-palvelut ovat oikeastaan jo ottaneet omakseen, on sisällön suoratoisto. Suoratoisto tarkoittaa median toistamista palvelimelta ilman, että sitä tarvitsee ensin kopioida paikallisesti levylle. Mediaa, esimerkiksi musiikkia, puskuroidaan tarvittava määrä muistiin, jotta sen toisto voidaan aloittaa. Lataus jatkuu toiston ohella, eikä kappaletta tallenneta paikallisesti levylle lainkaan. Asiakkaan palvelukokonaisuus voitaisiin tulevaisuudessa muuttaa suoratoistopohjaiseksi. Tekniikka suoratoistolle on olemassa, mutta monet taustamusiikkipalveluita tarvitsevista yrityksistä eivät ole vielä tarpeeksi nopean Internetyhteyden päässä, jotta suoratoisto onnistuisi riittävän luotettavasti. Lisäksi pienilläkin käyttäjämäärillä tietoliikenteen määrä olisi niin valtava, että suoratoistoa tarjoavien palvelimien sijoittaminen olisi hankalaa tai hyvin kallista.

6. YHTEENVETO

Taustamusiikkia soitetaan monenlaisissa tiloissa. Toistaminen hoidetaan digitaalisia järjestelmiä käyttäen ja pyritään automatisoimaan mahdollisimman tehokkaasti. Työn asiakasyritys on toimittanut yli kahdenkymmenen vuoden ajan taustamusiikkisoitinsovellusta erilaisiin yökerhoihin, ravintoloihin, ostoskeskuksiin ja baareihin. Diplomityön tavoitteena oli toimittaa asiakkaalle järjestelmä, jolla loppukäyttäjät voivat tilata musiikkia ja muuta mediaa, sekä hallita mediakirjastoaan ja asiakkuustietojaan. Työ tehtiin iteratiivisesti tiiviissä yhteistyössä asiakkaan kanssa. Projektinhallintamenetelmänä käytettiin Scrumia ja iteraatioiden, eli sprinttien pituudeksi valittiin kaksi viikkoa. Scrumia sovellettiin niiltä osin kuin siitä katsottiin olevan hyötyä, eli varsinaisesti rooleja ei nimetty pysyvästi ja sprintteihin otettiin lisää tehtäviä tuotelistalta kesken iteraatioiden. Toteutusta tekemässä oli yksi Scrum-tiimi, joka koostui projektin vaiheesta riippuen kahdesta viiteen henkilöstä. Projekti aloitettiin kesällä 2010 lähtien liikkeelle asiakkaan laatimasta määrittelystä ja alustavista käyttöliittymäkuvista.

Työhön sisältyi käyttöliittymän teko kuvien ja keskusteluiden perusteella, sovelluksen suunnittelu ja toteutus Java EE -ohjelmistoalustalle, yksikkötestien teko, järjestelmätestien teko, sekä yhdistetty ylläpito- ja suunnitteludokumentti. Jokaisen sprintin jälkeen asiakkaan kanssa pidettiin palaveri, jossa katsottiin kahden viikon aikaansaannokset, kirjattiin parannusehdotuksia ja muutostoiveita jo tehtyihin asioihin, ja samalla suunniteltiin mitä seuraavien kahden viikon aikana tehdään. Monet käyttöliittymäkomponentit ja toiminnot suunniteltiin näissä asiakkaan kanssa pidetyissä palavereissa. Toteutuksen ohessa järjestelmää testattiin kaikilla yleisimmillä selaimilla. Projektin loppupuolella asiakas lisäksi toimitti testikäyttöä varten monessa loppukäyttäjän kohteesta löytyvän pc-tietokoneen, jossa on kosketusnäyttö. Kosketusnäytön kanssa tehdystä testauksesta saatiinkin tuloksena tärkeitä huomioita ja parannusehdotuksia.

Haastavin osuus projektia oli ennalta laadittu tietomalli, johon ei saanut tehdä muutoksia. Tietokannan tuli säilyä ehyenä kaikkien sitä käyttävien järjestelmien näkökulmasta, mutta mitään tarkkoja ohjeita tai sääntöjä ei sovittu esimerkiksi kuinka mitäkin arvoja tulisi validoida. Haasteita aiheutti myös ajastustoiminto, joka päätettiin toteuttaa viikkokalenteria muistuttavana komponenttina. Kalenterikomponentin toteutus oli kuitenkin oletettua paljon työläämpi. Tästä huolimatta ajastustyökalu

on yksi onnistuneimmista toiminnoista järjestelmässä. Erityisen hyvin onnistui myös profiilieditori, joka on intuitiivinen käyttää ja sen ominaisuudet olivat lopulta suunniteltua laajemmat. Heikoimmin yksittäisistä toiminnoista onnistui soittolistaeditori, jonka toteutuksen tiedettiinkin olevan ongelmallista. HTML-spesifikaatiosta ja selaimista puuttuu raahaa ja pudota -toiminnallisuus, jonka toteutus web-palveluun on ongelmallista. Kolmannen osapuolen komponenttia käyttäen ominaisuus saatiin tehtyä, mutta se ei toimi esimerkiksi kosketusnäytöllä ollenkaan. Palveluun toteutettiin lisäksi toiminnot, joilla kappaleita voi lisätä ja poistaa soittolistalta ilman raahaa ja pudota -toiminnallisuutta.

Asiakkaan antama palaute järjestelmästä on ollut positiivista läpi projektin, ja projekti on ollut muillakin tavoilla mitattuna onnistunut. Projektin käyttöönotto on vielä kesken, mutta toteutuspuolella tuskin tulee enää isoja muutoksia. Käyttöönottoa ovat jarruttanut asiakkaan muiden järjestelmien päivitystyöt ja asiakasta koskeneet yrityskaupat. Järjestelmän jatkokehityksestä ja kokonaisuuteen liittyvistä muista palveluista on jo ollut puhetta asiakkaan kanssa.

LÄHTEET

[1] Apache Tomcat. Internet, 2011. http://tomcat.apache.org/. Sivuilla käyty viimeksi: 12.03.2011.

[2] Codd E.F. *A Relational Model of Data for Large Shared Data Banks.* Communications of the ACM, New York, USA 1970.

[3] Core J2EE Patterns - Data Access Object. Internet, 2011. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObje%ct.html. Sivuilla käyty viimeksi: 27.06.2011.

[4] CSS 2.1 Specification. Internet, 2010. http://www.w3.org/TR/CSS2/. Sivuilla käyty viimeksi: 13.03.2011.

[5] Dynamic HTML and XML: The XMLHttpRequest Object. Internet, 2005. http://developer.apple.com/internet/webcontent/xmlhttpreq.html. Sivuilla käyty viimeksi: 13.03.2011.

[6] EclipseLink. Internet, 2011. http://www.eclipse.org/org/press-release/20080317_Eclipselink.php. Sivuilla käyty viimeksi: 13.03.2011.

[7] ECMAScript Language Specification. Internet, 2009. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.p%df. Sivuilla käyty viimeksi: 13.03.2011.

[8] Fogie S. et al. *XSS Attacks: Cross Site Scripting Exploits and Defense.* Syngress Publishing, 2007.

[9] Fowler M. Inversion of Control. Internet, 2005. http://martinfowler.com/bliki/InversionOfControl.html. Sivuilla käyty viimeksi: 13.03.2011.

[10] GPLv2. Internet, 2011. http://www.gnu.org/licenses/old-licenses/lgpl-2.0.html. Sivuilla käyty viimeksi: 12.03.2011.

[11] Haikala I., Märijärvi J. *Ohjelmistotuotanto.* Talentum Media Oy, Jyväskylä 2006.

[12] HTML 4.01 Specification. Internet, 1999. http://www.w3.org/TR/html401/. Sivuilla käyty viimeksi: 13.03.2011.

[13] HTML 5 Working Draft. Internet, 2011. http://www.w3.org/TR/html5/. Sivuilla käyty viimeksi: 13.03.2011.

[14] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. 2000. 23 s.

[15] Java EE. Internet, 2011. `http://java.sun.com/j2ee/overview.html`. Sivuilla käyty viimeksi: 12.03.2011.

[16] Java Expression Language. Internet, 2011. `http://download.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html`. Sivuilla käyty viimeksi: 13.03.2011.

[17] JavaScript Object Notation. Internet, 2011. `http://www.json.org/`. Sivuilla käyty viimeksi: 13.03.2011.

[18] JAX Innovation Award GeWinner. Internet, 2006. `http://jax.de/`. Sivuilla käyty viimeksi: 13.03.2011.

[19] Jenkins Continuous Integration. Internet, 2011. `http://jenkins-ci.org/`. Sivuilla käyty viimeksi: 22.04.2011.

[20] Jetty. Internet, 2011. `http://www.eclipse.org/jetty/`. Sivuilla käyty viimeksi: 12.03.2011.

[21] Johnson R. *Expert One-on-One J2EE Design & Development*. Wrox Press Ltd., Birmingham, UK, UK, 2002.

[22] Jolt awards. Internet, 2006. `http://drdobbs.com/architecture-and-design/187900423?pgno=10`. Sivuilla käyty viimeksi: 13.03.2011.

[23] jQuery JavaScript Library. Internet, 2011. `http://jquery.com/`. Sivuilla käyty viimeksi: 13.03.2011.

[24] JSR 220. Internet, 2011. `http://jcp.org/en/jsr/detail?id=220`. Sivuilla käyty viimeksi: 12.03.2011.

[25] Koskimies K., Mikkonen T. *Ohjelmistoarkkitehtuurit*. Talentum Media Oy, Jyväskylä 2005.

[26] Moore D. et al. Inferring Internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24:115–139, May 2006.

[27] MySQL. Internet, 2011. `http://www.mysql.com/`. Sivuilla käyty viimeksi: 22.06.2011.

[28] Nevalainen R. Ketterästi tehtyä ja laadukasta - onnistuuko? Internet, 2008. http://www.pcuf.fi/sytyke/lehti/kirj/st20084/ST084-18A.pdf. Sivuilla käyty viimeksi: 19.06.2011.

[29] Nielsen J. Usability testing with 5 users. Internet, 2000. http://www.useit.com/alertbox/20000319.html. Sivuilla käyty viimeksi: 09.09.2011.

[30] Nielsen J. 10 Heuristics for User Interface Design. Internet, 2005. http://www.useit.com/papers/heuristic/heuristic_list.html. Sivuilla käyty viimeksi: 09.09.2011.

[31] Oppel A. *Databases Demystified.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2004.

[32] RFC 2616 - HTTP/1.1. Internet, 1999. http://tools.ietf.org/html/rfc2616. Sivuilla käyty viimeksi: 22.04.2011.

[33] Scrum. Internet, 2011. http://www.scrumalliance.org/. Sivuilla käyty viimeksi: 02.06.2011.

[34] Selenium web application testing system. Internet, 2011. http://seleniumhq.org/. Sivuilla käyty viimeksi: 22.04.2011.

[35] Spolsky J. Joel on exceptions. Internet, 2003. http://www.joelonsoftware.com/items/2003/10/13.html. Sivuilla käyty viimeksi: 25.04.2011.

[36] Spool J. et al. *Web Site Usability: A Designer's Guide.* User Interface Engineering, 1997.

[37] Spring Framework. Internet, 2011. http://www.springsource.org/. Sivuilla käyty viimeksi: 13.03.2011.

[38] Stripes. Internet, 2011. http://www.stripesframework.org/. Sivuilla käyty viimeksi: 13.03.2011.

[39] TestNG testing framework. Internet, 2011. http://testng.org/. Sivuilla käyty viimeksi: 22.04.2011.

[40] Vincit - uusien ideoiden ehdoilla. Internet, 2011. http://www.vincit.fi/asiakkaille/kertovat. Sivuilla käyty viimeksi: 20.06.2011.

[41] W3C Document Object Model. Internet, 2005. http://www.w3.org/DOM/. Sivuilla käyty viimeksi: 13.03.2011.